UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Gº en Ingeniería en Informática

Trabajo final del Grado en Ingeniería Informática:

Aplicabilidad de Gazebo en la simulación de vehículos autónomos

Presentado por Carmelo Basconcillos Reyero
en julio de 2017
Tutor D. Jesús Enrique Sierra García

Índice de contenido

Índice de ilustraciones

Índice de tablas

I - PLANIFICACIÓN

1. Introducción

Se va a ver como ha ido evolucionando el proyecto durante su realización, así como un estudio de viabilidad respecto a cómo se puede llevar evar a cabo en el futuro. Para la realización de este proyecto hemos seguido la metodología *Scrum*, la cual consiste en tener una estrategia de desarrollo incremental del producto y semanalmente juntar varias fases en paralelo del proyecto. En la decisión no se ha optado por una metodología *Scrum* ya que asi en cada fase vamos dando valor añadido al proyecto, siempre con versiones nuevas a través de las cuales nuestro trabajo va adquiriendo un valor añadido. Además de ir marcándonos objetivos semanalmente.

La metodología semanal ha sido la siguiente:

- Abrimos *Git-Hub*
- Creamos una ***Milestone*** con duración de una semana (generalmente).
- Vamos creando *issues* correspondientes a las tareas, de tal manera que las asociamos a un *milestone*, un tiempo estimado, un autor, y un proyecto.
- Respecto a la gestión temporal se realiza desde ZenHub, que es una herramienta incluida en el navegador e integrada en GitHub, las tareas se pasan de nuevas a abiertas. Esto también facilitará gráficas sobre este trabajo.
- Según se va finalizando las tareas, se van llevando a tareas cerradas. Aquí nos cierra el *issue*, y se podrá ver el gráfico que indica el progreso perfecto frente al progreso real.

2. Planificación temporal

El proyecto se ha llevado a cabo marcando objetivos semanales e intentando cumplirlos. Como veremos a continuación, alguna semana en la que se disponía de más tiempo, se ha incluido alguna tarea extra y por el contrario en otra, se ha dejado una determinada tarea retrasada.

A continuación se plasma el trabajo realizado cada semana:

2.1. Sprint 1 (6/2/2017 – 19/2/2017)

Objetivos:	Aprender el uso del repositorio, crear máquina virtual y entorno de trabajo
Tareas realizadas:	Es una toma de contacto respecto a la gran cantidad de herramientas con las que se va a trabajar. Lo primero y fundamental es la creación de una máquina virtual *Ubuntu 16.04*. Sobre ella se va a instalar el software de *Gazebo*. A continuación, se seguirán distintos tutoriales encontrados en Internet. El principal inconveniente es que el primero lleva una versión de *ubuntu* más antigua, y ralentiza mucho la máquina virtual. Tras analizar algunas herramientas de gestión de proyectos como *Git, Bitbucket o GitHub*, se decide trabajar con GitHub como gestor de versiones y con *ZenHub* como gestor de proyectos Gracias a esto se aprende el uso del repositorio, mediante nuestro entorno de trabajo, para más adelante realizar diferentes tutoriales de la página oficial de gazebo tanto de su manejo como del modelado de robots.
Cumplido:	No se sabe si se ha cumplido el objetivo, dado que no se conoce como usar el repositorio inicialmente, teniendo que cambiar los milestones para que generen gráficas. Aún con esto, el imprevisto de reinstalar la máquina virtual habría retrasado casi con seguridad.

Tabla 1: Sprint 1

2.2. Sprint 2 (20/2/2017 – 26/2/2017)

Objetivos:	Realizar el primer modelo de robot en Gazebo
Tareas Realizadas:	Para ello se ha hecho un primer diseño en papel de lo que se quería (aconsejados por el tutor), se ha realizado diferentes tutoriales de SDF y se ha llevado a cabo la realización y puesta en movimiento del mismo.
Cumplido:	Esta semana se ha dado un gran paso, puesto que se ha entendido como funcionan los robots, donde y como crear las librerías para que nuestros robots aparezcan en el programa, y se ha sabido manejar bien gazebo.

Tabla 2: Sprint 2

2.3. Sprint 3 (27/2/2017 – 12/3/2017)

Objetivos:	Informarse y documentarse sobre ROS. Instalación en el sistema
Tareas Realizadas:	Después de un fallo con la versión de *ubuntu*, se ha conseguido instalar ROS con todas las librerías kinetic y ambién se ha visto el uso del manejador de fuerzas Rviz. A su vez se ha realizado los primeros tutoriales de ROS, en los que se ha aprendido como funciona ROS, sistema publicador/suscriptor.
Cumplido:	Se han cumplido los objetivos, aunque la realización de los tutoriales ha agotado más tiempo de lo previsto, ya que que en ocasiones aparecen errores, no compilan ciertas partes,... La `primera semana en la que se permite rescatar la gráfica de trabajo y conocer la realización de tutoriales, fue una carga muy pesada. Esto se aprecia en que las tareas se realizan muy al final.

Tabla 3: Sprint 3

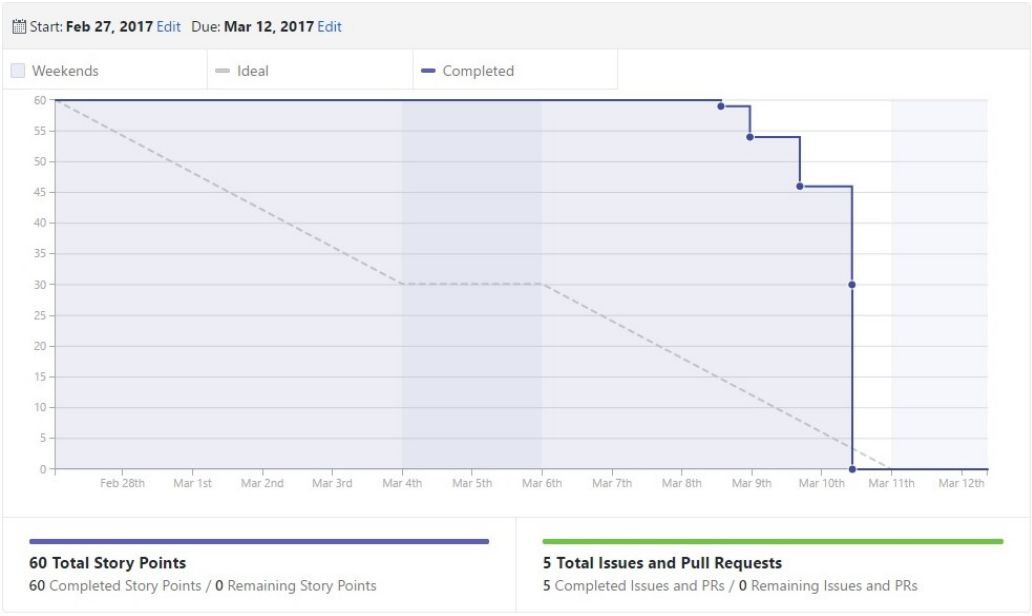


Ilustración 1: A.1 Burndown del Sprint 3

2.4. Sprint 4 (13/3/2017 - 19/3/2017)

Objetivos:	Lanzar el primer robot con ROS
Tareas Realizadas:	Esta semana es la de los primeros pasos reales en ROS. Se ha conseguido lanzar un mundo con un robot el cual, en el momento que se lanza es capaz de escribir un “Hola mundo”. Es la primera piedra sobre la que se sostenta ROS, el *publisher* **(**donde publica los mensajes cada robot).
Cumplido:	Se han cumplido los objetivos y además los tiempos marcados son muy similares a la linea de la gráfica perfecta, por lo que está semana se acaba muy contentos con el trabajo realizado.

Tabla 4: Sprint 4

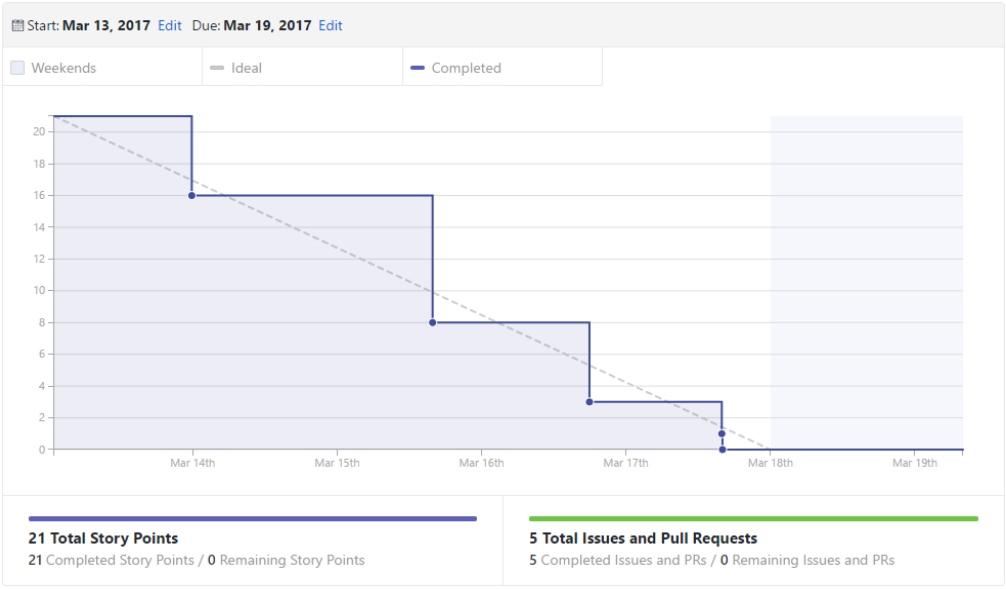


Ilustración 2: A.2 Burndown del Sprint 4

2.5. Sprint 5 (20/3/2017 - 26/3/2017)

Objetivos:	Robot interprete ordenes
Tareas Realizadas:	Se han dado los pasos previos para que el robot interprete órdenes, reciba y mande mensajes por medio de ROS, un objetivo intermedio, aunque no el que nos marcamos al inicio de la semana.
Cumplido:	En esta semana no se ha conseguido cumplir los objetivos, lo que si que se ha logrado es realizar los tutoriales que había previstos. Aunque no se ha conseguido el objetivo puesto que el robot no interpreta ordenes todavía, se dan los primeros pasos para lograrlo en un futuro y se deja para siguientes. La curva de trabajo vuelve a ser casi idónea.

Tabla 5: Sprint 5

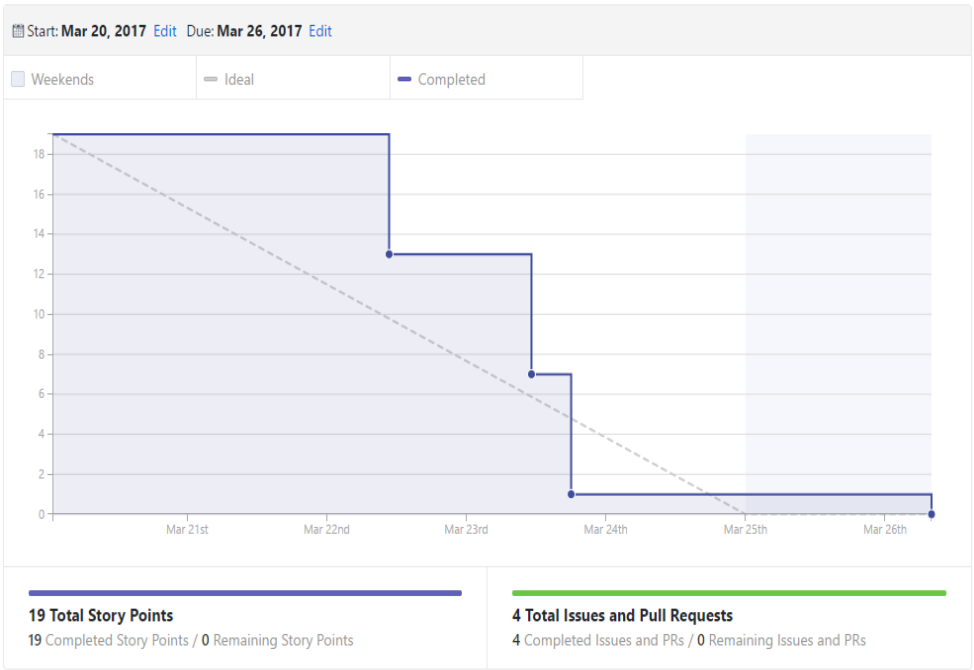


Ilustración 3: A.3 Burndown del Sprint 5

2.6. Sprint 6 (27/3/2017 - 2/4/2017)

Objetivos:	Realizar tutoriales que permitan mover el robot
Tareas Realizadas:	En la semana pasada ya se consiguió que el robot robot escuchase y hablase. En este caso se ha probado con un *jackal* y con un *turtlebot* (dos modelos de robots de la librería Gazebo). Para más posteriormente insertar estos comandos a nuestros robots. Se han realizado los tutoriales para ver si se aprende sobre como hacer que el robot interprete ordenes, de momento sin éxito.
Cumplido:	Se han realizado los tutoriales y se ha comprendido mejor como funcionan, pero aún sin ver como hacer que un robot lanzado por nosotros interprete ordenes.

Tabla 6: Sprint 6

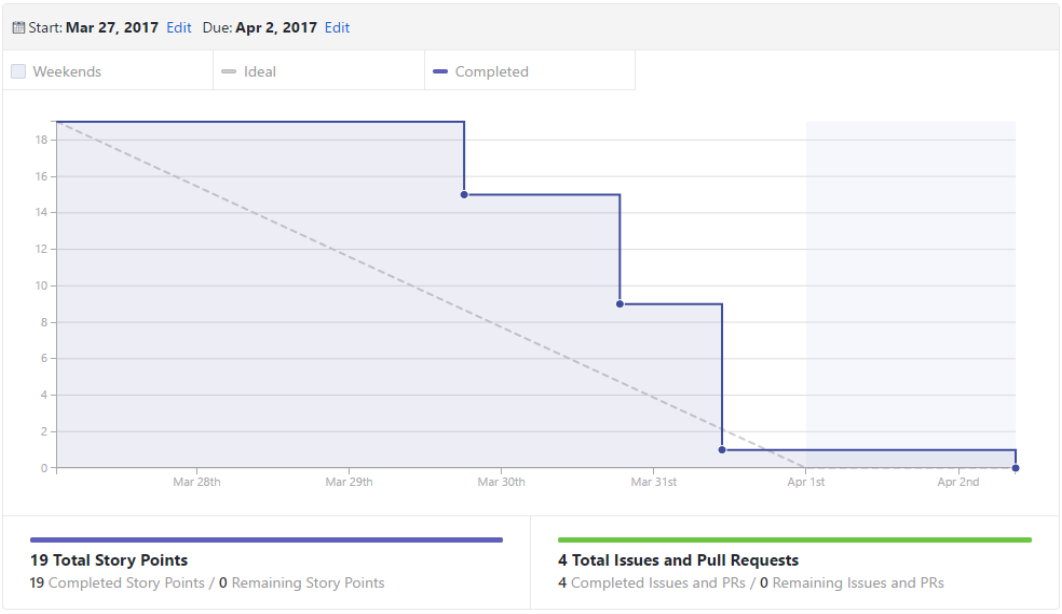


Ilustración 4: A.4 Burndown del Sprint 6

2.7. Sprint 7 (3/4/2017 - 9/4/2017)

Objetivos:	Mover nuestro robot
Tareas Realizadas:	Se ha intentado insertar movimiento desde ROS, programando ordenes para dar fuerza a las uniones de las ruedas de nuestro robot, pero después de perder mucho tiempo no se ha conseguido que el robot se mueva por medio de ROS.
Cumplido:	Esta semana no se ha cumplido el objetivo. Aparecen fallos en el modelado y faltan parámetros de velocidad en las uniones para poder insertarle velocidad desde ROS. Además no ha funcionado lo que se quería por culpa de esto, por lo tanto la gráfica de trabajo ha quedado retrasada porque no se han cumplido los plazos.

Tabla 7: Sprint 7

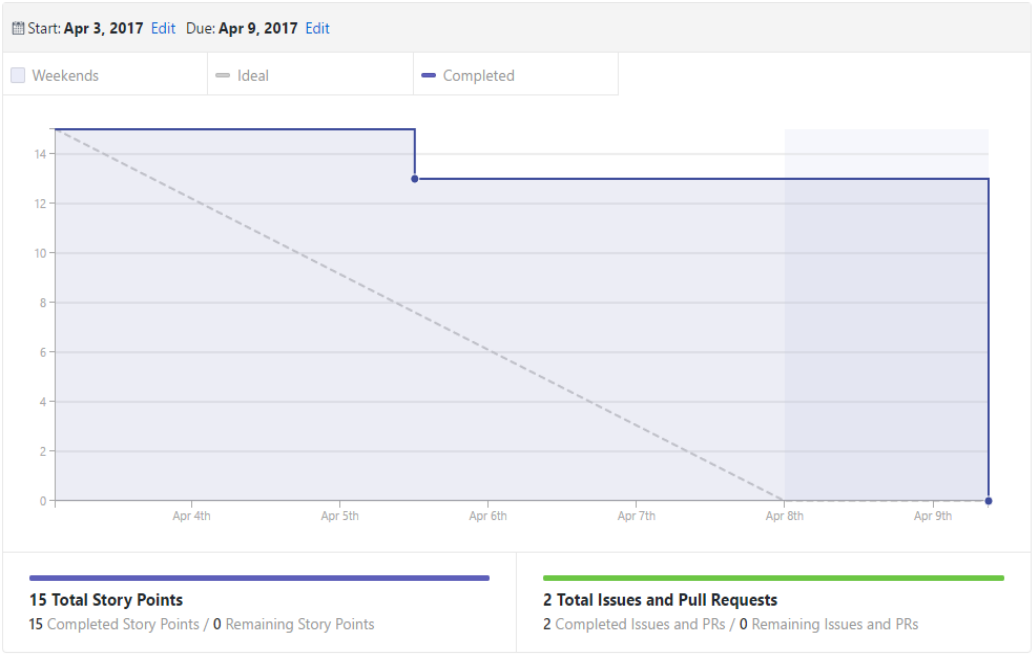


Ilustración 5: A.5 Burndown del Sprint 7

2.8. Sprint 8 (17/4/2017 - 23/4/2017)

Objetivos:	Corregir fallos del modelado, mover el robot
Tareas Realizadas:	Se ha conseguido que el robot sea capaz de interpretar ordenes de movimiento, de tal manera que pueda mover sus uniones.
Cumplido:	Esta semana se conseguirá el objetivo, si bien los plazos de tiempo como se observa en la gráfica son tardíos, puesto que finalmente dedicamos más tiempo a ello y logramos conseguir el movimiento en el robot por medio de ROS.

Tabla 8: Sprint 8

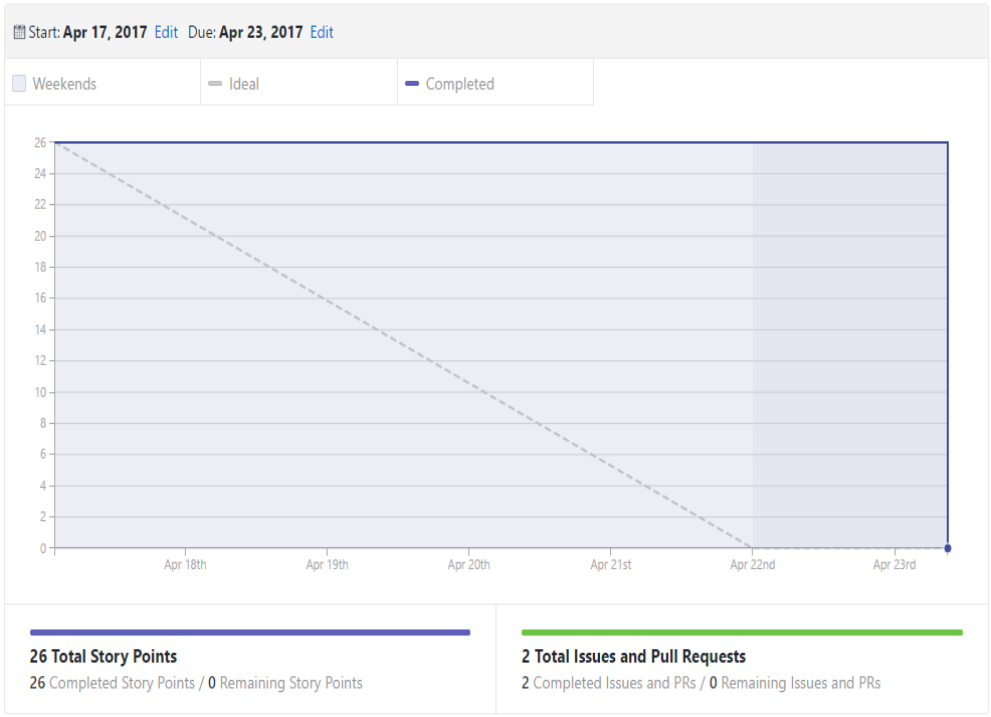


Ilustración 6: A.6 Burndown del Sprint 8

2.9. Sprint 9 (24/4/2017 - 30/4/2017)

Objetivos:	Realizar manual de usuario
Tareas Realizadas:	Se empezará la realización del manual de usuario que más tarde se entregará como anexo a la memoria (uno de los objetivos del proyecto para reducir la costosa curva de aprendizaje de ROS).
Cumplido:	Manual de usuario completado de manera eficiente, con esto se cumpliría uno de los objetivos del proyecto. La caída de la curva es muy pronunciada en el gráfico porque solo teníamos una tarea grande, no vale de mucho en este caso el gráfico.

Tabla 9: Sprint 9

Ilustración 7: A.7 Burndown del Sprint 9

2.10. Sprint 10 (1/5/2017 - 7/5/2017)

Objetivos:	Realizar memoria
Tareas Realizadas:	Semana que se dedica expresamente a hacer memoria para que quede una gráfica más real. Se han cogido todos los puntos de la memoria que se ha ido realizando y los que se han hecho issues, así se podrá llevar un mejor seguimiento del tiempo. Se han realizado las partes de objetivos del proyecto mediante conceptos teóricos e introducción.
Cumplido:	La propuesta fue de en dos semanas de trabajo completar la memoria, y se ha llegado a hacer la mitad más o menos, por lo cual el objetivo está cumplido.

Tabla 10: Sprint 10

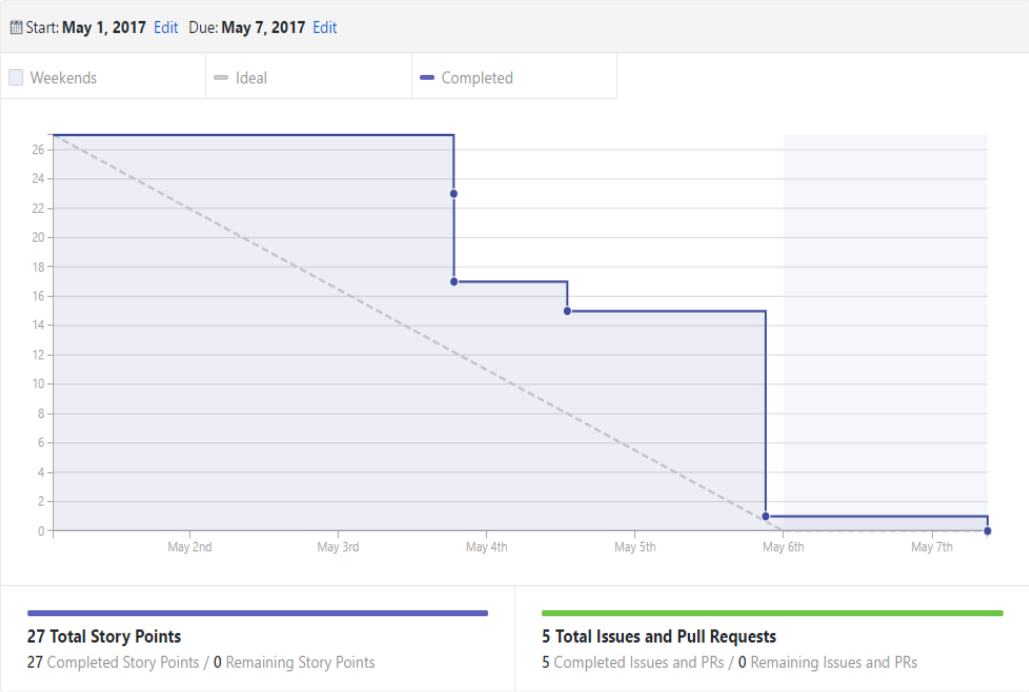


Ilustración 8: A.8 Burndown del Sprint 10

2.11. Sprint 11 (8/5/2017 - 14/5/2017)

Objetivos:	Fijar puntos del informe y realizar los primeros de ellos
Tareas Realizadas:	En esta semana se quiere empezar lo que sería el informe a entregar. Después de hablar con el Tutor se han acordado los puntos y se han llevado a cabo los apartados de estudio de mercado, la planificación inicial y los recursos.
Cumplido:	La reunión fue satisfactoria, salió todo adelante y se obtiene la primera parte del entregable prácticamente finalizada. Por lo cual el objetivo está cumplido.

Tabla 11: Sprint 11

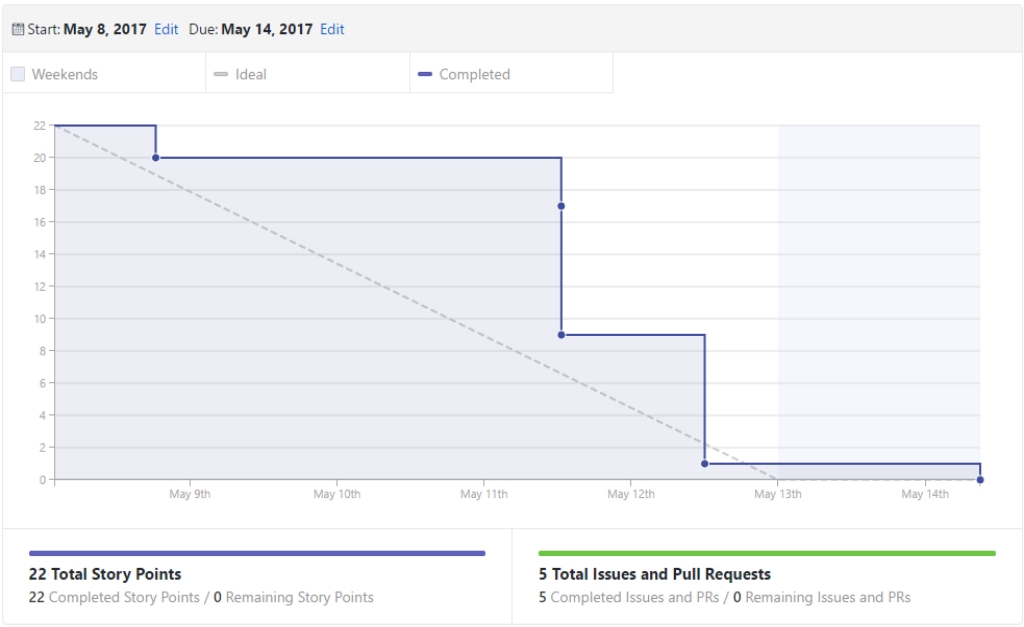


Ilustración 9: A.9 Burndown del Sprint 11

2.12. Sprint 12 (15/5/2017 - 21/5/2017)

Objetivos:	Finalizar el informe.
Tareas Realizadas:	Se ha realizado el plan financiero, puntos de aprendizaje y la cronología de las tareas del informe, pero se puede dar por terminado.
Cumplido:	No se ha llegado a completar el informe, pero se han conseguido puntos muy importantes a seguir en él, trabajo constante en la semana, aunque sin llegar al objetivo final.

Tabla 12: Sprint 12

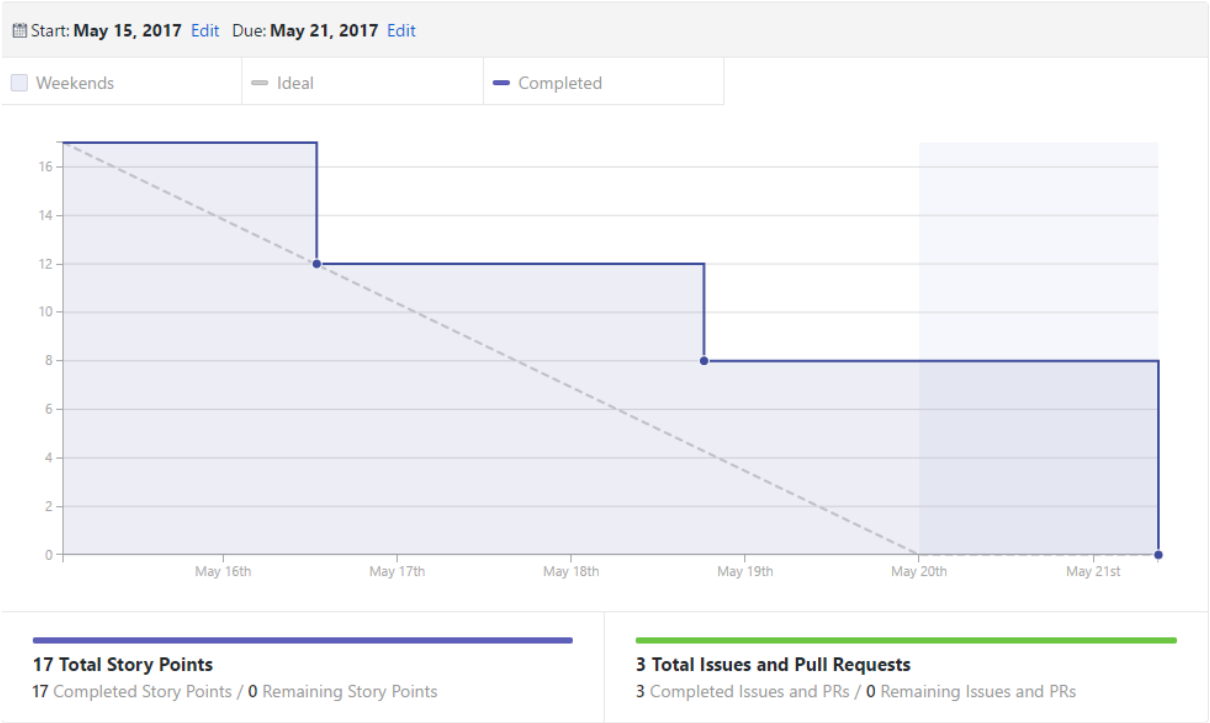


Ilustración 10: A.10 Burndown del Sprint 12

2.13. Sprint 13 (22/5/2017 - 28/5/2017)

Objetivos:	Finalizar la memoria
Tareas Realizadas:	Se han realizado los puntos referentes a las técnicas y herramientas, aspectos relevantes, trabajos relacionados y conclusión de la memoria
Cumplido:	Se ha conseguido acabar la memoria, el trabajo ha sido bueno y constante, tal y como muestra el gráfico.

Tabla 13: Sprint 13

Ilustración 11: A.11 Burndown del Sprint 13

2.14. Sprint 14 (1/6/2017 – 30/6/2017)

Objetivos:	Finalizar anexos, realizar curso, crear *moodle*, realizar vídeos
Tareas Realizadas:	En estas 4 semanas se han realizado todos los puntos importantes para finalizar el proyecto. Llevando a cabo la finalización de anexos, realizando el curso, creando videotutoriales y subiéndolos a un moodle que ha sido previamente creado.
Cumplido:	Se ha conseguido acabar correctamente todo el trabajo que se pretendía.

Tabla 14: Sprint 14

Ilustración 12: Burndown del Sprint 14

Tal y como se aprecia en la siguiente gráfica, en las últimas 8 semanas de trabajo (de las primeras no guardé información) se aprecia un trabajo semanal constante. Donde la media es de 22 horas por semana, lo que son sin contar los fines de semana unas 4,5 horas al día con el proyecto. El último sprint es de 4 semanas por eso tiene un pico en la gráfica más elevado.

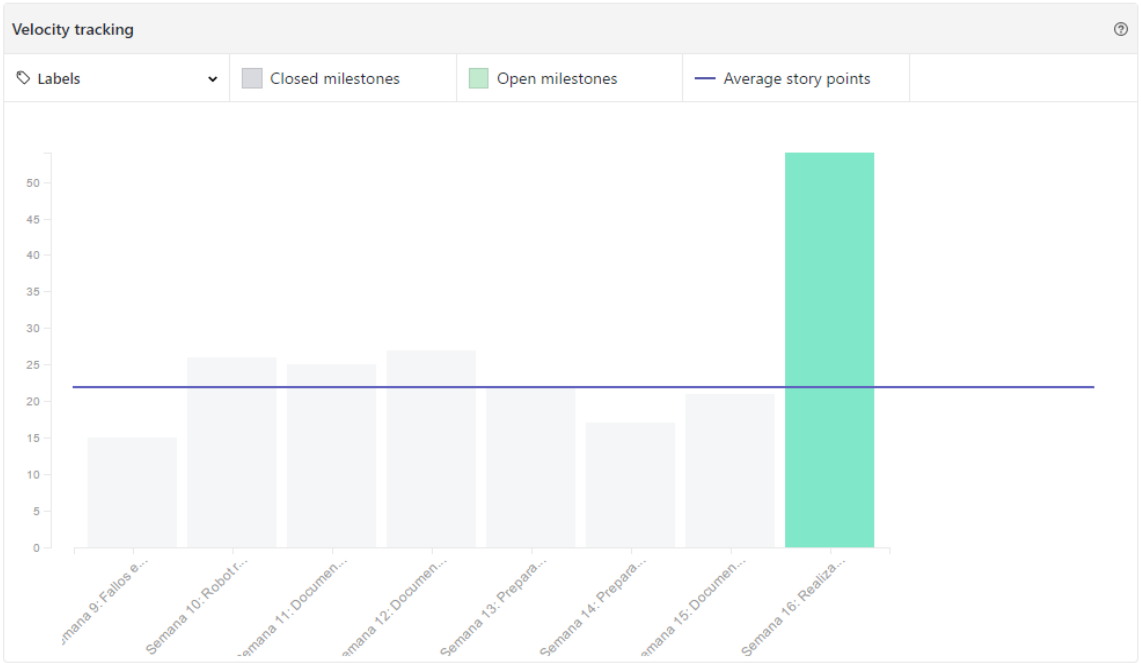


Ilustración 13: A.12 Horas invertidas por semanas

2.15. Sprint 15(16/8/2017-18/9/2017)

Objetivos:	Eliminar curso, quitar partes de la memoria y anexos innecesarias, mejorar la redacción, realizar pruebas, hacer vídeos de las diferentes demostraciones.
Tareas Realizadas:	En estas 4 semanas el trabajo realizado ha consistido en mejorar, modificar o añadir partes del trabajo que faltaban o que estaban erróneas, así como dejar una buena documentación en la máquina virtual de todo el trabajo realizado. Han sido una media de 3 horas al día durante estas 4 semanas.
Cumplido:	Se ha conseguido acabar todo el proyecto por lo que podemos decir que está última semana ha sido un éxito.

Tabla 15: Sprint 15

3. Estudio de viabilidad

En esta sección se va a analizar si se puede apostar o no por este proyecto en un futuro. Se tendrán en cuenta diferentes puntos de vista, así como la viabilidad económica y legal. Sin duda ambos factores son determinantes para poder seguir con el proyecto.

3.1. Costes personal

Para un único trabajador durante 5 meses y medio, podemos usar las horas invertidas en *github* para hacer con más exactitud los cálculos:

Es necesario contratar a un trabajador con experiencia en C++ y C, que tenga cierta soltura para el aprendizaje de nuevos lenguajes (como SDF). Estimamos que el sueldo de un programador senior será de 26000€ brutos anuales, por lo que si 2017 ha tenido 225 días laborables (quitando 24 días de vacaciones), sale a un precio de 115,55€ por día trabajado. Cada día tiene 8 horas de trabajo, por lo que sale a una media de 14,44€ por hora.

Teniendo en cuenta que se han usado 544 horas* 14,44€/hora = 7855.36€ paga la empresa por el trabajador.

3.2. Costes de material

El software utilizado no tiene coste ya que todo lo que usamos no necesita licencias de pago. En el caso de *VMware* su coste de licencia es de 251,95€, aunque será usada una versión no comercial para el proyecto pero si se quiere comercializar habría que pagar esta licencia.

Por el hardware yo he usado mi propio equipo, cuyas características son Intel Core i7 8 Gb de RAM y un disco duro de 1TB, que tiene un valor aproximado de 900€. Se considera que la amortización de un equipo ronda los 5 años y lo he usado 5 meses y medio por lo tanto el coste en hardware ha sido:

900€ / (12Meses * 5 años) = 15€/mes

5,5 Meses de uso * 15€/mes = 82,5€

Los gastos mensuales de Internet son de 35€ mensuales, suponiendo 175€ al finalizar el proyecto.

Costes	Importe
Personal	7855.36€
Material	82,5€
Internet	175€
Licencias	251,95€
Total:	8364,81€

Tabla 16: Tabla de costes

3.3. Viabilidad legal

En este punto lo que se va a hacer es analizar las libreras que se han usado en el proyecto. Es importante buscar los tipos de licencias que tienen cada uno y examinarlas dependiendo de las compatibilidades. Hay que conocer qué tipos de licencia se pueden aplicar y en este caso las que se han utilizado son:

ROS: BSD.

SDF: Licencia Apache 2.0.

Vmware: Licencia de Pago. Una alternativa es Virtual Box de licencia GPL.

Ubuntu: GPL y otras licencias libres.

Gazebo: Licencia Apache 2.0.

Como vamos a nombrar *copyleft*, es necesario explicar brevemente de lo que trata. Consiste en permitir una libre distribución de copias y versiones modificadas, donde los derechos de autor persisten en las versiones modificadas. [1]

La licencia GNU GPL es la licencia más utilizada en cuanto a software libre. Tiene como propósito decir que el software es libre, pero se encuentra protegido mediante copyleft, de apropiación externa.[2]

La licencia de ROS es una licencia BSD de software libre pero que no permite copyleft a diferencia de la GNU GPL. La licencia BSD no es compatible con la licencia GPL. [3]

En cuanto a la licencia Apache, se trata también de una licencia de software libre permisiva, requiere la conservación de los derechos de autor pero no es copyleft.[4]

Una vez realizado este análisis de las licencias utilizadas, se ha decidido poner una licencia GNU GPL al proyecto dado que queremos mantener los derechos de autor, además gracias al copyleft mantenemos los derechos si se llevan a cabo nuevas versiones.

BSD de tres clausulas (También llamada nueva BSD) es compatible con GPL, pero la licencia mas restrictiva es la GPL, por lo que nuestro software deberá llevar dicha licencia.

II - ESPECIFICACIÓN DE REQUISITOS

1. Introducción

El objetivo de este apartado es la definición de las características que se desea que la aplicación tenga al final de su desarrollo. Para ello se fijarán los objetivos que se intentan cumplir y, más formalmente, se definirán los requisitos funcionales y no funcionales de la aplicación. Finalmente se profundizará en los casos de uso derivados de la funcionalidad que se desea.

2. Objetivos generales

Tal y como se trató en la memoria, el proyecto cuenta con dos objetivos claramente definidos.

Principalmente el objetivo es aprender a usar esta tecnología, para más adelante conseguir realizar un informe donde se saquen unas buenas conclusiones, y se consiga diseñar un curso online con el cual la implantación de esta tecnología sea mucho más sencilla.

3. Catalogo de requisitos

RF-1: Modelar nuestro robot dentro de la plataforma Gazebo

RF-1: Obtener un primer diseño del robot

RF-2: Identificar sus partes

RF-3: Modelar el diseño a ordenador poniendo los tipos de objeto en función a su utilidad

RF-4: Probar el funcionamiento del robot en la plataforma Gazebo para su correcto funcionamiento

RF-2: Conectarnos con el robot creado por medio de ROS

RF-1: Crear un mundo con sus elementos esenciales

RF-2: Lanzar nuestro robot dentro del mundo

RF-3: Lanzar nodos suscriptor y publicador

RF-4: Realizar programación para movimiento del robot

RF-5: Almacenar ordenes en scritpts con movimientos

RF3: Diseñar un informe de viabilidad para una empresa

RF-1: Redactar los puntos importantes del informe

RF-2: Documentar todos los puntos en función a los conocimientos adquiridos

RF-3: Visualizar informe

RF4: Elaborar pruebas

RF-1: Detectar las pruebas que podrían ser útiles

RF-2: Realizar la programación de las pruebas

RF-3: Realizar vídeo explicativo

4. Especificación de requisitos

4.1. Diagrama de casos de uso

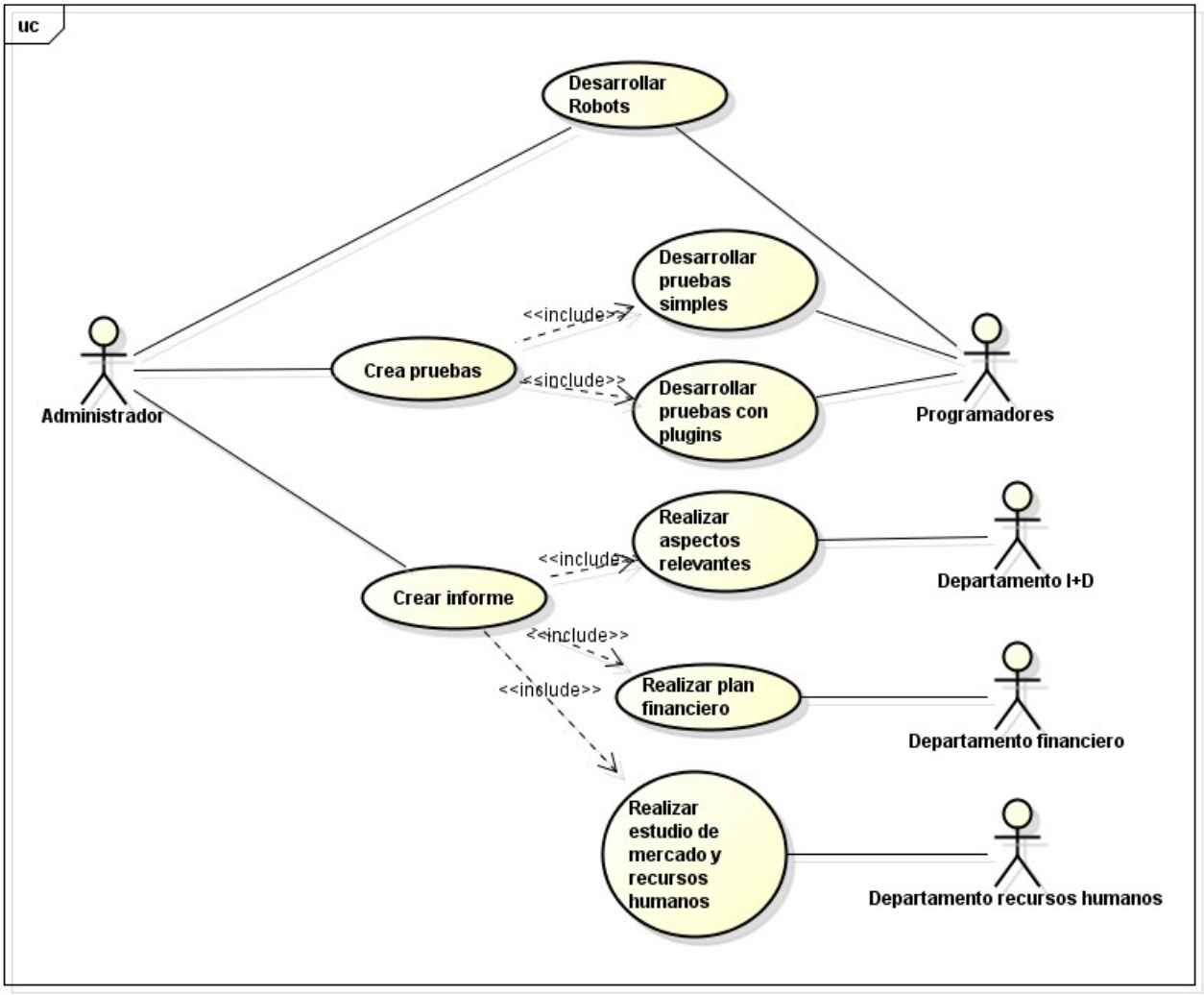


Ilustración 14: Diagrama de casos de uso

En la ilustración 14, se observa el diagrama de casos de uso del curso. A continuación vamos a ver más detalladamente cada caso de uso. Son cinco los actores que se pueden encontrar en el diagrama de casos de uso. El administrador se ocupa de realizar todas las acciones, y el usuario únicamente puede registrarse y hacer acciones referentes a realizar el curso.

Caso de uso	Desarrollar robots
Requisitos	RF-1 RF-1.1 RF-1.2 RF-1.3 RF-1.4
Descripción	Desarrollar diferentes modelos de robots y entornos
Precondiciones	Se debe haber aprendido el lenguaje de modelo y funcionamiento del simulador
Acciones	1. Se diseña un modelo en papel 2. Se modela en el simulador 3. Se comprueba su funcionamiento
Postcondiciones	Si el robot se comporta de manera defectuosa revisarlo, realizar pruebas
Excepciones	Pueden desarrollarse entornos en vez de robots
Importancia	Alta

Tabla 17: Desarrollar Robots

Caso de uso	Crear pruebas
Requisitos	RF-1,4 RF-2 RF-4 RF-4.1 RF-4.2 RF-4.3
Descripción	Se deben de crear pruebas sobre los diferentes modelos
Precondiciones	Debemos de haber desarrollado anteriormente los robots
Acciones	1. Desarrollar entorno 2. Lanzar mundo 3. Realizar la prueba
Postcondiciones	Si la prueba resulta fallida, revisar para encontrar fallos
Excepciones	Ninguna
Importancia	Media

Tabla 18: Crear pruebas

Caso de uso	Desarrollar pruebas simples
Requisitos	RF-1.3 RF-1.4 RF-2.1 RF-2.2 RF-4 RF-4.1 RF-4.2 RF-4.3
Descripción	Desarrollar diferentes pruebas simples
Precondiciones	Debemos de haber desarrollado anteriormente los robots
Acciones	1. Desarrollar entorno 2. Lanzar mundo 3. Buscar objetivo de la prueba 4. Realizar la prueba
Postcondiciones	Si la prueba resulta fallida, revisar para encontrar fallos
Excepciones	Ninguna
Importancia	Media

Tabla 19: Desarrollar pruebas simples

Caso de uso	Desarrollar pruebas con plugins
Requisitos	RF-2,1 RF-2,2 RF-2.3 RF-2.4 RF-2.5 RF-4.1 RF-4.2 RF-4.3
Descripción	Desarrollar diferentes pruebas con plugins de librería de movimiento
Precondiciones	Tenemos que tener generada la librería, hecha la compilación
Acciones	1. Cargar librerias 2. Lanzar nodo master 3. Lanzar nodo que ejecute movimientos 4. Evaluar la prueba
Postcondiciones	Si la prueba resulta fallida, revisar para encontrar fallos
Excepciones	Puede dar fallos al lanzar las pruebas
Importancia	Alta

Tabla 20: Desarrollar purebas con plugins

Caso de uso	Crear informe
Requisitos	RF-3 RF-3.1 RF-3.2 RF-3.3
Descripción	Crear informe de Gazebo
Precondiciones	Haber realizado todas las pruebas, y desarrollado todos los robots
Acciones	1. Fijar los puntos a desarrollar 2. Redactar puntos 3. Realizar previsión financiera 4. Realizar conclusión y opinión final
Postcondiciones	Ninguna
Excepciones	Ninguna
Importancia	Muy alta

Tabla 21: Crear informe

Caso de uso	Realizar aspectos relevantes
Requisitos	RF-3 RF-3.2 RF3.3
Descripción	Realizar los puntos del informe como objetivos, temporalidad, metodología...
Precondiciones	Haber realizado todas las pruebas, y desarrollado todos los robots
Acciones	1. Describir los puntos 2. Realizar el estudio de los puntos 3. Rellenar informe
Postcondiciones	Ninguna
Excepciones	Ninguna
Importancia	Alta

Tabla 22: Realizar aspectos relevantes

Caso de uso	Realizar plan financiero
Requisitos	RF-3 RF-3.2 RF3.3
Descripción	Realizar plan financiero de implantar estas técnicas
Precondiciones	Documentarte sobre los datos de la empresa
Acciones	1. Realizar una previsión de gastos 2. Realizar una previsión de ingresos 3. Realizar un plan de amortización
Postcondiciones	Revisar los datos, puesto que el mercado avanza diariamente
Excepciones	Ninguna
Importancia	Alta

Tabla 23: Realizar plan financiero

Caso de uso	Realizar estudio de mercado y recursos humanos
Requisitos	RF-3 RF-3.2 RF3.3
Descripción	Realizar un estudio de mercado y recursos humanos
Precondiciones	Documentarse sobre empresas competidoras
Acciones	1. Realizar estudio de mercado fijandote en otras empresas del sector y su tecnología en torno a los simuladores 2. Realizar una previsión de personas para trabajar en este proyecto
Postcondiciones	Ninguna
Excepciones	Ninguna
Importancia	Media

Tabla 24: Realizar estudio de mercado y recursos humanos

III - ESPECIFICACIÓN DE DISEÑO

1. Introducción

En este apartado se va a explicar qué se ha usado para la resolución del problema en cuanto a herramientas de diseño.

Para la elaboración de los diagramas de paquetes y de clases se ha usado UML como lenguaje de modelado y el programa *Astah* para implementar los diagramas de clases.

2. Diseño de datos

En la ilustración número 16 podemos ver el diagrama de clases de nuestro código. La clase ros es una librería de ros, que no ha sido creada por mí pero si se definen atributos y métodos de esta clase dado que se incluyen esta librería en el *Listner*.

Listner es quien se encarga de la comunicación con ROS.

Comandos es la función intermedia que comunica lo recibido en el subscriptor de ROS y nos lanza el comando a la clase MiRobot que interpreta la orden.

La clase common y physics son dos clases de Gazebo de las cuales hereda MiRobot, en MiRobot se producen las ordenes al robot para que se mueva.

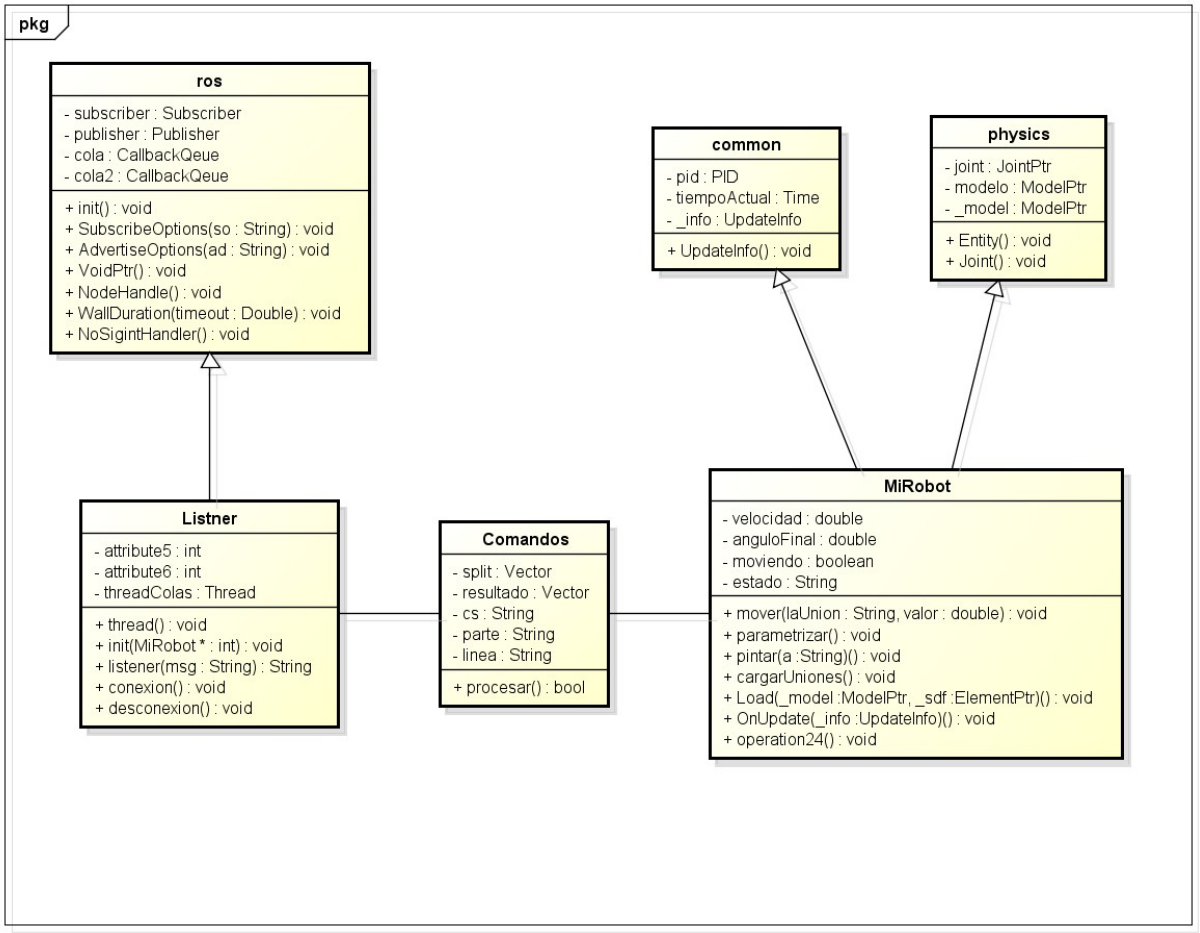


Ilustración 15: Diagrama de clases

3. Diseño procedimental

En cuanto al diseño procedimental será basado en el código de ejemplo que se ha creado para el movimiento del robot, y para una posible simulación. Se puede ver en la ilustración 17 como se llaman las clases entre si.

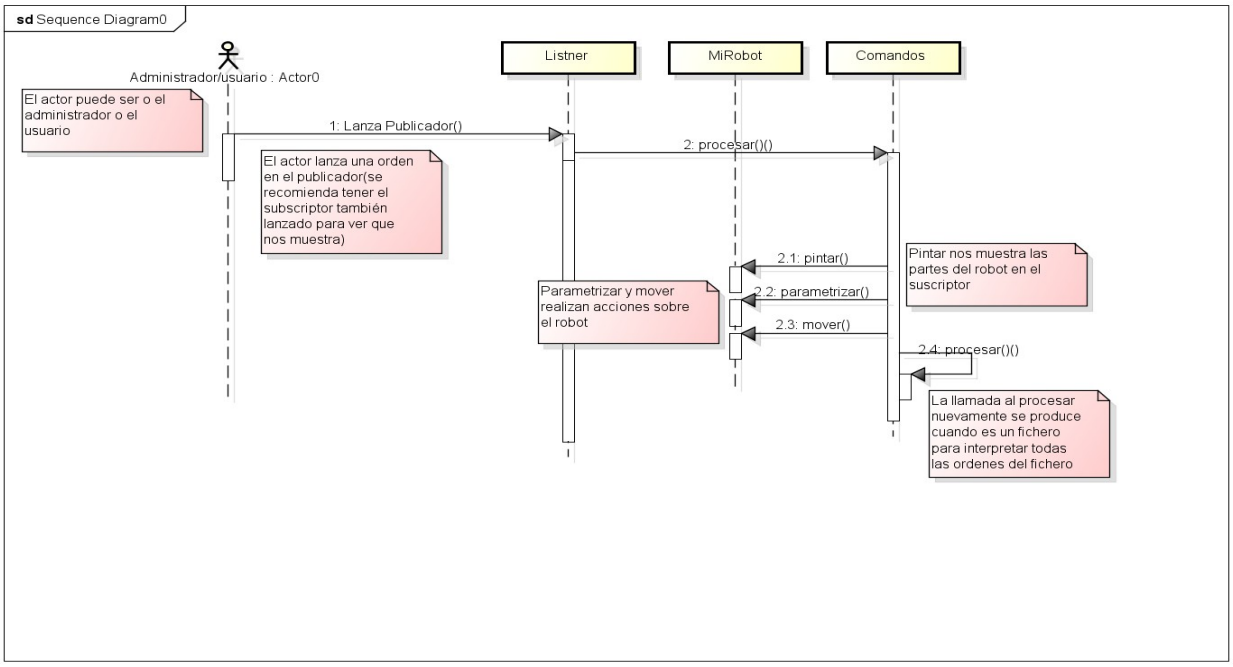


Ilustración 16: Diagrama de secuencias de nuestro código

En la parte del informe se ha realizado un informe analítico, que tiene como objetivo justificar una decisión ya proyectada. Es una justificación de la viabilidad de esta herramienta en empresas del sector. [5]

IV - Documentación técnica de programación

1. Introducción

Esta sección esta dirigida a otros desarrolladores, de modo que puedan continuar el proyecto y entenderlo. En el se describen en detalle el funcionamiento del proyecto y los aspectos que podrán mejorarse o modificarse en el futuro.

2. Estructura de directorios

Existen tres carpetas principales en el proyecto:

- **Carpeta Devel:** La carpeta devel es la ubicación por defecto donde los ejecutables y librerías van antes de ser instaladas.
- **Carpeta bin:** contiene los ejecutables del paquete, los que se podrán lanzar como nodos.
- **Carpeta lib:** contiene los archivos de librerías (.lib, .so). Aquí se crea la librería de comunicación con el robot cuando se compila el código (se encuentra dentro de Devel.)
- **Build:** Contiene los archivos de compilación internos del paquete.
- **Src:** Contendrá los códigos de los programas que se creen (.cpp) y estos al compilarlos crearán ejecutables en bin.
- **Carpeta worlds:** Contiene el mundo (definición de gravedad, suelo, luz...) con nuestro robot (.world). Dentro de src.
- **Carpeta launcher:** La carpeta launcher contiene el lanzador, el que instanciamos primero para crear nuestro mundo. Dentro de worlds.
- **Archivo package.xml:** Proporciona meta información acerca del paquete.
- **Archivo CmakeList:** Archivo que usa *catkin* para compilar, es una lista en la que se especifica al compilador que debe compilar de nuestro paquete: nodos, mensajes, servicios, librerías, etc.
- **Archivo manifest.xml:** contiene una descripción del paquete (función, autor, licencia, etc.) y una lista con los paquetes de los que depende. Según la funcionalidad del paquete, es posible encontrar más o menos carpetas con archivos diferentes. En el caso más común de un paquete que se usa para simulación en Gazebo.

```
workspace_folder/         -- WORKSPACE
  src/                    -- SOURCE SPACE
    CMakeLists.txt        -- The 'toplevel' CMake file
    package_1/
      CMakeLists.txt
      package.xml
      ...
    package_n/
      CMakeLists.txt
      package.xml
      ...
  build/                  -- BUILD SPACE
    CATKIN_IGNORE         -- Keeps catkin from walking this directory
  devel/                  -- DEVELOPMENT SPACE (set by CATKIN_DEVEL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...
  install/                -- INSTALL SPACE (set by CMAKE_INSTALL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...
```

Ilustración 17: Ejemplo paquete Catkin

3. Manual del programador

En este apartado se verán diferentes acciones que van a permitir un uso más avanzado del trabajo desarrollado.

3.1. Importar máquina virtual

Por desgracia el programa *VMware Player* carece de la funcionalidad de crear, copiar o exportar máquinas virtuales, pero se ha encontrado una solución que va a permitir clonar la máquina virtual creada.

Lo que se va a hacer es realizar literalmente una copia de los archivos que forman la máquina virtual.

Se buscan las máquina virtuales, en este caso C:\Users\Carmelo\Documents\Virtual Machines y copiamos todos los archivos de la máquina virtual del proyecto (se llama TFG).

En la siguiente imagen se ven todos los archivos que contiene.

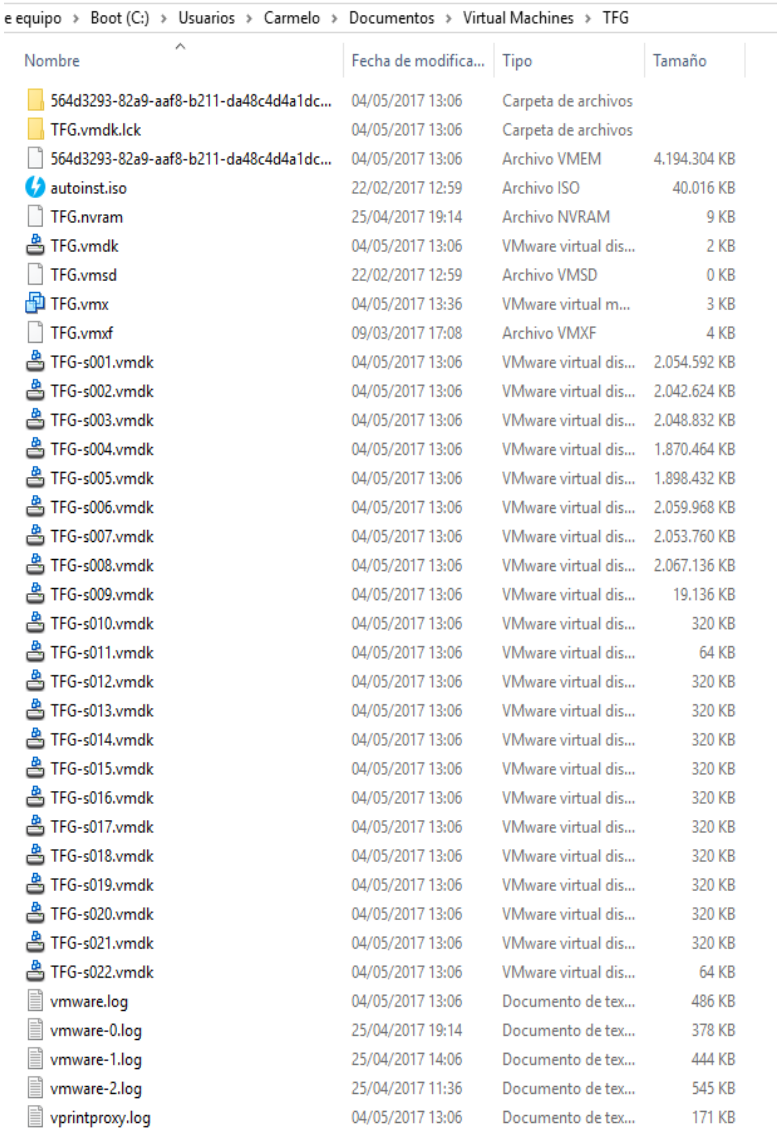


Ilustración 18: Todos los archivos de nuestra máquina virtual

Una vez “clonada” la máquina virtual, hay que llevar todos estos archivos a otro equipo donde importarla. En este nuevo equipo se arranca el *VMware Player* y se clica sobre *Open a Virtual Machine* como en la siguiente imagen.

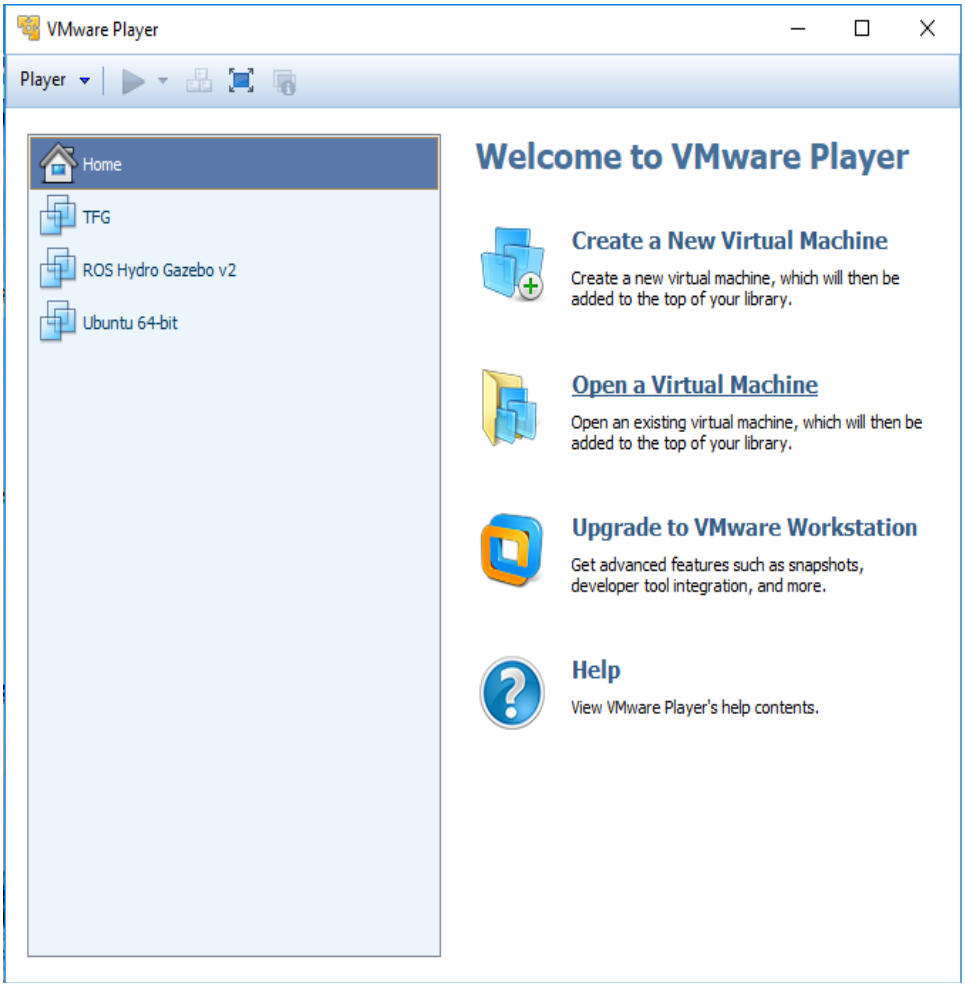
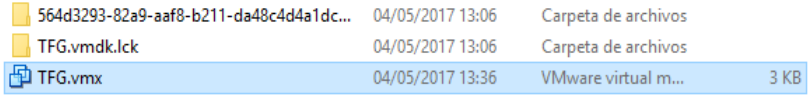


Ilustración 19: VMware Player

Se el fichero de configuración (.vmx) de la máquina virtual, se selecciona y se pulsa Abrir.



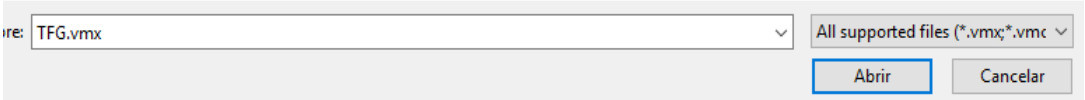


Ilustración 20: Importar máquina virtual

Una vez preparada la maquina virtual para su uso, solo hará falta ejecutarla.

Para poder usar la sesión con el entorno preparado hay que abrir la sesión que se ha usado con nombre Carmelo, y con contraseña “123456”.

3.2. Instalación de herramientas

3.2.A. Instalación de Gazebo

En la máquina virtual ya vienen instaladas las siguientes herramientas, pero se va a dejar constancia de como han sido instaladas.

Para la instalación de gazebo tan sólo hay que abrir un terminal y ejecutar el siguiente comando, además se instalará la versión más reciente del simulador.

```
curl -sSL http://get.gazebosim.org | sh
```

Ilustración 21: Instalación de Gazebo

Para ejecutar el programa se puede hacer de dos maneras, o bien cachearlo en el buscador de elementos de ubuntu, o bien abriendo un terminal y escribiendo gazebo.

3.2.B. Instalación de ROS

Para la instalación de ROS kinetic hay que tener en cuenta que solo soporta Ubuntu 15.10, Ubuntu 16.04 y Debian 8.

Primeramente hay que configurar el ordenador para aceptar software de ROS:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $ (lsb_release -sc) main"> /etc/apt/sources.list.
d/ros-latest.list'
```

Ilustración 22: Configurar ROS

Después deberemos instalar las claves:

```
sudo apt-key adv --sistema de servidor hkp: //ha.pool.sks-keyservers.net: 80 --recv-key 421
C365BD9FF1F717815A3895523BAEEB01FA116
```

Ilustración 23: Claves ROS

Se actualizarán todos los paquetes para que no haya problemas de compatibilidades de versiones.

```
sudo apt-get update
```

Ilustración 24: Actualizar

.Y más tarde se eligirá que versión de ROS se desea instalar, En este caso la versión completa, puesto que contiene complementos como rviz o simuladores que serán usados en nuestro proyecto.

```
sudo apt-get instalar ros-kinetic-desktop-full
```

Ilustración 25: Instalar ROS

Ya estaría instalado propiamente ROS en su versión kinetic, pero se necesita inicializar *rosdep*, para instalar en el sistemas dependencias y para poder utilizar algunas funciones básicas de ROS.

```
sudo rosdep init
actualización rosdep
```

Ilustración 26: Inicializar rosdep

3.3. Desarrollo y compilación

En esta sección vamos a describir como se han ido desarrollando los diferentes robots, y como se han ido desarrollando las diferentes pruebas.

3.3.A. Desarrollo de robots

Para desarrollar los robots el procedimiento ha sido:

- Desarrollar cada parte del robot
- Describir las uniones con cada parte del robot
- Dar valores como la velocidad máxima de uniones, altura máxima de la unión, fricción, máximo esfuerzo...
- Dar valores a la superficie de amortiguación, rigidez, contacto máximo de velocidad...

Con estas premisas se han desarrollado 3 robots, cada uno para un tipo de pruebas. Al construir los robots deben de ir siempre dentro de una etiqueta model, y todas las características del robot se incluirán dentro. En la ilustración 21 se aprecia como sería un ejemplo de construcción de un robot, donde se daría su posición inicial en el plano(pose), y se declararían las partes(link), dentro de cada link se deben de dar los elementos visuales y de contacto de esa pieza, así como datos de fricción, velocidad maxima, rigidez o amortiguación de esa pieza.

La etiqueta joint debe de llevar un tipo asociado según el tipo de unión, y determinará que partes usas, llevará parámetros de velocidad y dirá en que dirección(x y z) se ejerce la fuerza que se ejerce sobre esa unión.

La etiqueta plugin llevará dentro una librería de ROS creada con anterioridad, y que el robot incluirá, será su programación.

Se pueden incluir modelos creados con la etiqueta include, de esta manera se podrán añadir las cámaras o sensores que desees.

Example

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <model name="box">
    <pose>0 0 0.5 0 0 0</pose>
    <static>false</static>

    <link name="link">
      ...
    </link>

    <joint type="revolute" name="my_joint">
      ...
    </joint>

    <plugin filename="libMyPlugin.so" name="my_plugin"/>

  </model>
</sdf>
```

Ilustración 27: Ejemplo de modelo

Para incluir estos robots en la librería de gazebo tan solo se debe copiar el fichero y pegarlo en donde se encuentran el resto de robots C://Gazebo/models.

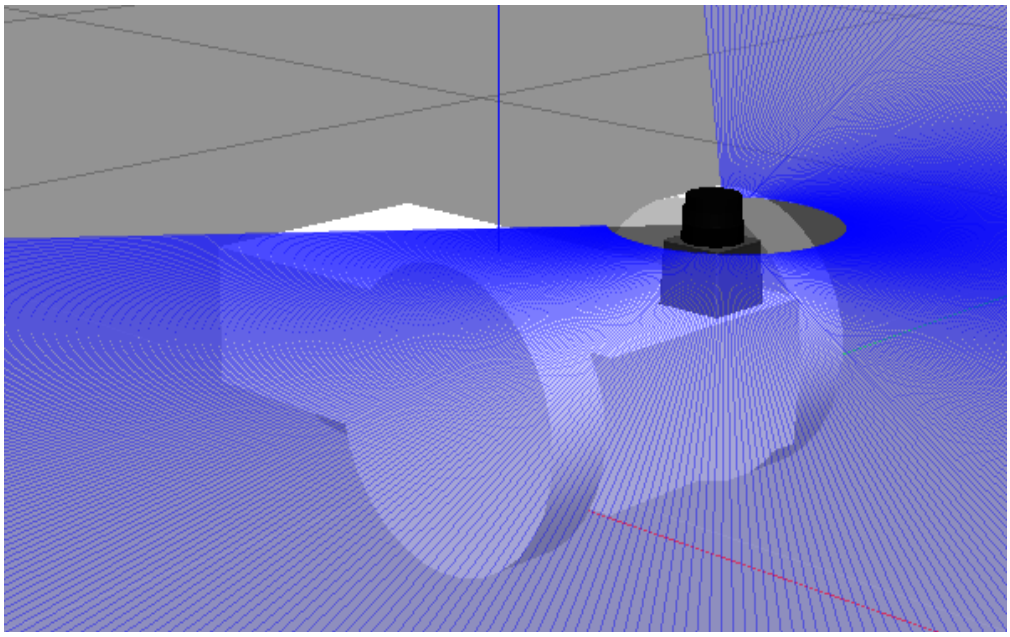


Ilustración 28: Modelo robot 1

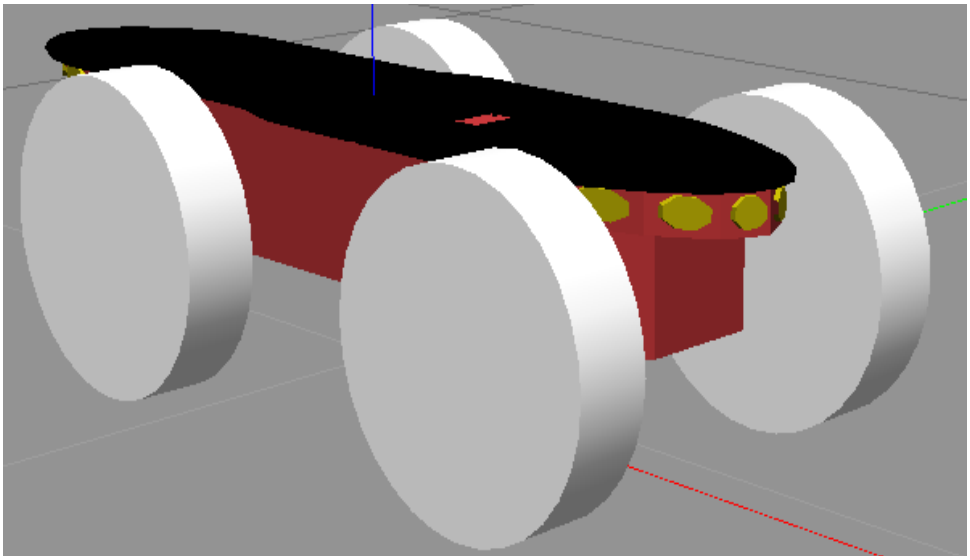


Ilustración 29: Modelo robot 2

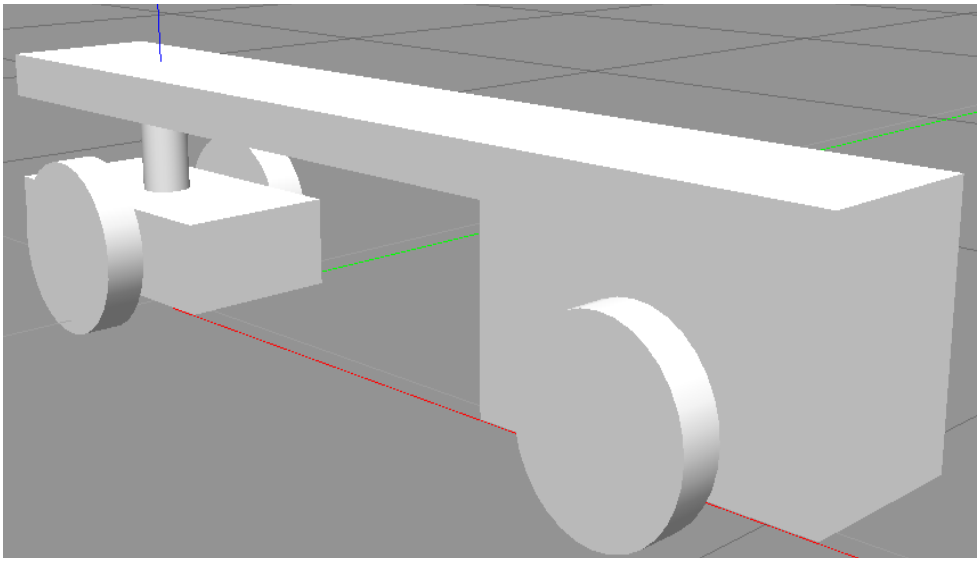


Ilustración 30: Modelo robot 3

3.3.B. Desarrollo de entorno

Para el desarrollo de las pruebas se ha necesitado la programación de los robots anteriormente detallados, y el desarrollo de unos entornos determinados que nos permitan llevarlos a cabo (se detalla mejor en los vídeos adjuntados en la entrega). Para desarrollar un mundo hay que detallar unos parámetros determinados, como la luz, la gravedad o el plano, vemos un ejemplo en la ilustración 25.

Example

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <world name="default">
    <physics type="ode">
      ...
    </physics>

    <scene>
      ...
    </scene>

    <model name="box">
      ...
    </model>

    <model name="sphere">
      ...
    </model>

    <light name="spotlight">
      ...
    </light>

  </world>
</sdf>
```

Ilustración 31: Ejemplo Mundo

Con esto tendríamos un mundo normal en el que lanzar los robots, pero si se quiere construir entornos reales como edificaciones o casas hay que usar el building editor de Gazebo.

Para ello hay que dirigirse al menú edit y se pulsa *Building Editor*.

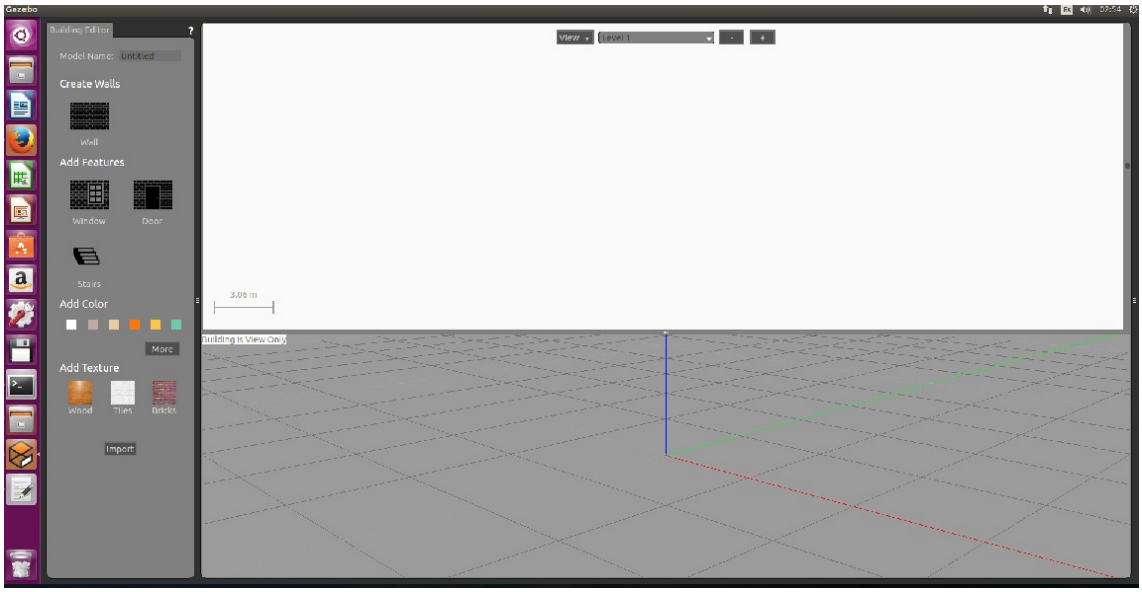


Ilustración 32: Building Editor

Ahora pulsamos import y seleccionamos la imagen de un plano que queramos desarrollar en Gazebo, seleccionamos una pared y elegimos los metros que tiene esa pared, asi gazebo sabra la escala que tiene que usar, y pulsamos ok.

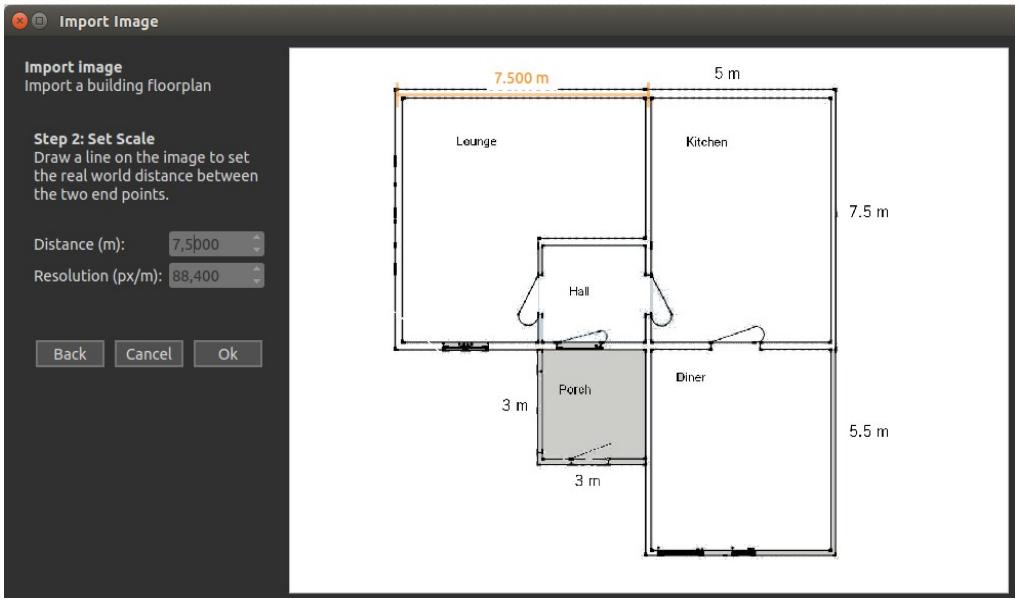


Ilustración 33: Plano

Ahora con el panel de la izquierda marcamos arriba las paredes(wall), y más tarde las ventadas y puertas(window y door), de tal manera que cada vez que marcamos en nuestro plano una pared puerta o ventana, nos lo va creando en la vista de debajo del simulador.

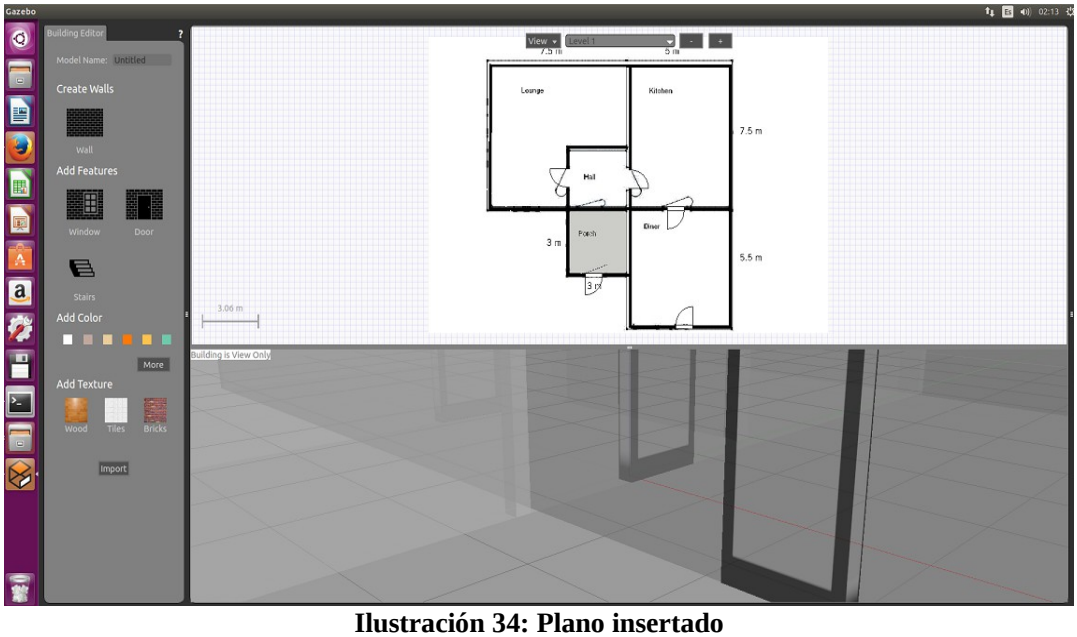


Ilustración 34: Plano insertado

Para terminar se seleccionan los colores que se desean en el menú de la izquierda para decorar las paredes, y ya se obtendría la casa diseñada.

3.3.C. Desarrollo de pruebas

Para su compilación la única manera de comprobar si su desarrollo era correcto era por medio de pruebas, se incluían los robots en el simulador, y se aplicaban fuerzas para comprobar que su comportamiento era correcto.

Para las pruebas que se realicen por medio del código facilitado por Raúl Trapiela para mover a los robots por medio de ROS, es necesario compilar cada parte del código de la siguiente manera.

Hay que abrir un terminal, y siempre que se abre uno hay que cargar las librerías de ROS para que reconozca los comandos que posteriormente serán introducidos. Para ello se usa la siguiente secuencia:

```
carmelo00@ubuntu:~$ source robot/devel/setup.bash
carmelo00@ubuntu:~$
```

Ilustración 35: Cargar librerias

Para compilar el código creado se deberá acceder a la carpeta donde están los paquetes. En este caso *mirobot*, y ejecutar el siguiente comando.

```
carmelo00@ubuntu:~$ cd robot
carmelo00@ubuntu:~/robot$ catkin_make --force-cmake
```

Ilustración 36: Compilar

Para poder ver las fuerzas que se ejercen sobre estos robots se ha usado el visualizar Rviz, que viene incluido en el paquete que se ha instalado de Gazebo. Para lanzarla hay que tener el ros-node activo, y más tarde lanzarla por medio de:

```
$ rosrun rviz rviz
```

Seleccionar el botón de abajo a la izquierda que dice Add:

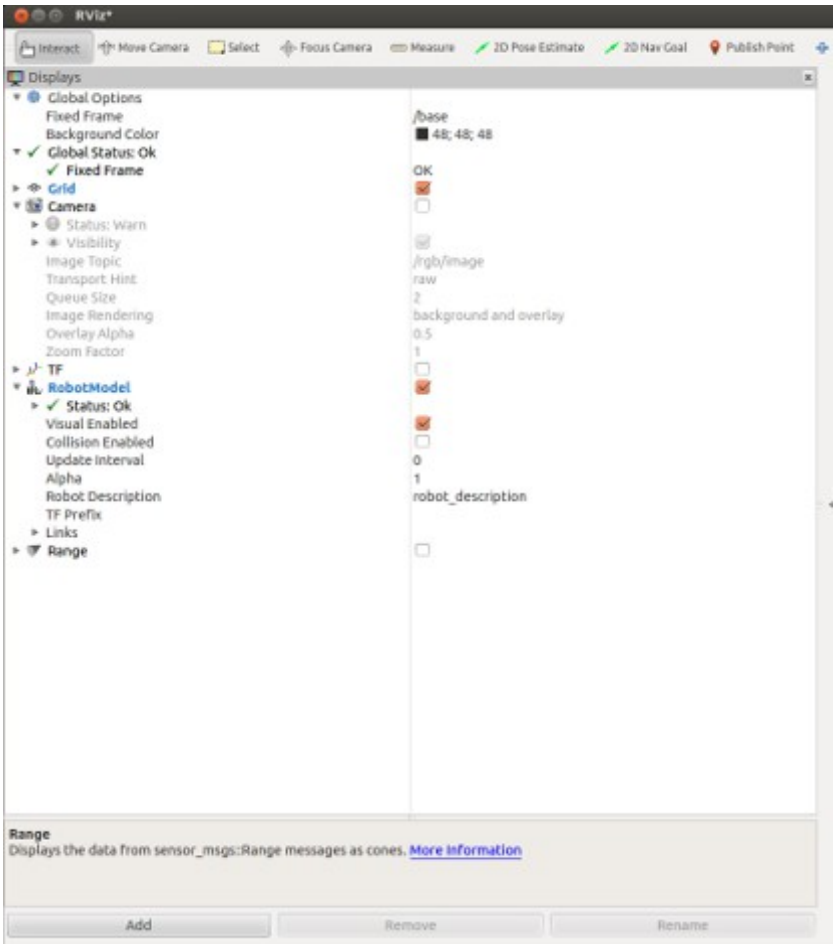


Ilustración 37: Rviz

Y se elige el robotmodel que hay creado.

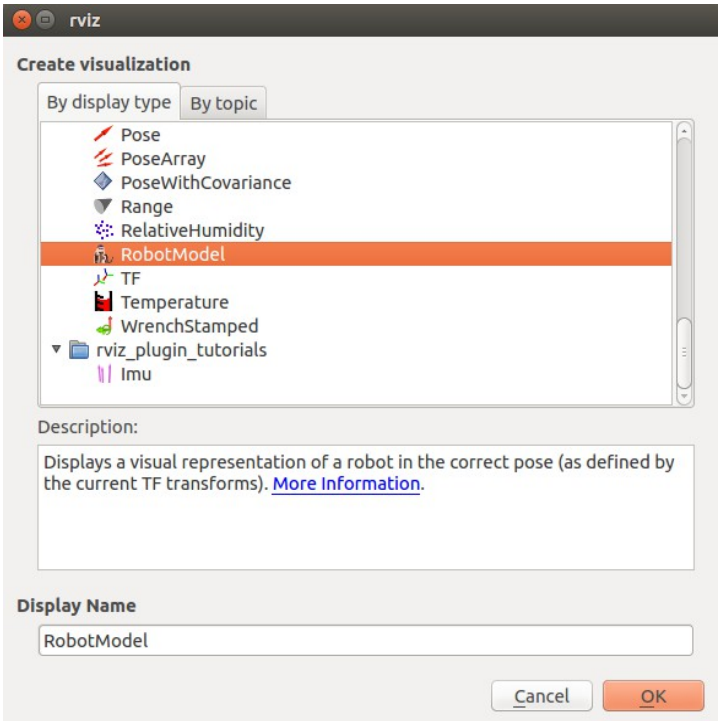


Ilustración 38: Añadir Rviz

3.4. Versiones de fallos

Fallo/Errores	Descripción
Primera instalación en kubuntu	Fue la primera versión para pruebas con el simulador Gazebo, pero era un sistema muy limitado con el que trabajar
Máquina virtual de matlab	Se prueba una máquina virtual de la ayuda del programa matlab, pero estaba desactualizada en la versión 6 de gazebo, por lo que pronto es deshechada
Versión ROS	Se prueba con la versión Hydro de ROS, pero al darse cuenta de que no tenia algunas funciones que aparecian en los manuales de ROS, se intenta con la versión kinetic
Sobrecalientamiento PC	El simulador Gazebo gasta muchos recursos del sistema, y llega a quemar la tarjeta de video de la placa base.
Métodos de librería Gazebo	Algunos de las referencias a métodos de las librerías de Gazebo no funcionan correctamente
Mostrar parámetros de sensor Hokuyo	No se consigue que muestre los parámetros de este sensor, pese a seguir varias indicaciones de diversos tutoriales.

Tabla 25: Tabla de fallos

Por estos errores ya citados se aconseja seguir las instrucciones en las versiones que se aconsejan, dado que sino podrían aparecer numerosos fallos.

V - DOCUMENTACIÓN DE USUARIO

1. Introducción

Esta sección se centra en la explicación y demostración de que características hay que tener para poder acceder y realizar el curso.

2. Requisitos de usuarios

Esta sección tiene como objetivo indicar aquellos requisitos que necesita el usuario para poder utilizar el proyecto.

2.1. Requisitos software

Es necesario tener instalado el gestor de máquinas virtuales VMWare Player. Este gestor puede ser utilizado prácticamente en cualquier sistema operativo, soportando la mayoría de sistemas Windows, Linux, Mac OS X y Solaris. Dado que no se puede clonar, luego explicaremos bien como la se ha copiado, para más tarde pegarla en otro ordenador.

2.2. Requisitos hardware

En la parte hardware seremos un poco exigentes, necesitamos que pueda correr en la máquina virtual sin problemas. Necesitamos mínimo una RAM de 2GB en la máquina virtual, y un disco duro de al menos 20 GB, el procesador del equipo a poder ser que sea potente, en nuestro caso usamos un I7 a 2,2Ghz y debe usarse uno igual o superior.

3. Manual de usuario

3.1. Primeras pruebas

En esta sección se explicará como el usuario puede ejecutar las pruebas una vez importada la máquina virtual en su equipo(se explica detalladamente en la sección manual del programador como hacerlo).

Para ejecutar las pruebas será muy sencillo, las primeras 4 pruebas se realizan ejecutando los 4 primeros lanzadores que tenemos en la máquina virtual, de tal manera que:

- Superar un cuerpo se corresponde con prueba1.launcher

Modelo de robot	Modelo de 2 ruedas
Objetivo	Impactar con un cuerpo y superar un objeto y ver como se comporta en la caída el robot.
Resultado	El robot se comporta de forma adecuada, cae bien, y sigue con su trayectoria, aunque se aprecia un ligero desvío después del choque. También con este modelo podemos ver como se comporta el láser Hokuyo que hemos insertado a este modelo.

Tabla 26: Prueba 1

- Impactar con cuerpo pesado se corresponde con prueba2.launcher

Modelo de robot	Modelo de 4 ruedas
Objetivo	Impactar con un cuerpo, y ver como actúa aumentando su velocidad
Resultado	El robot choca con un cuerpo muy pesado, y es capaz de tirarlo, pero no de superarlo, se aumenta su velocidad para que consiga pasarlo y continuar con su trayectoria después del impacto

Tabla 27: Prueba 2

- Un mundo sin suelo se corresponde con prueba3.launcher

Modelo de robot	Modelo AGV
Objetivo	Ver comportamiento de un mundo mal programado
Resultado	Se puede observar como el agv al introducirse en el simulador se precipita al vacío al no tener un suelo para apoyarse

Tabla 28: Prueba 3

- Muestra un nuevo entorno complejo se corresponde con prueba4.launcher

Modelo de robot	Sin modelo
Objetivo	Ver un entorno real
Resultado	Se puede visualizar un entorno en el cual se ha desarrollado una casa con sus paredes, puertas y ventanas

Tabla 29: Prueba 4

De tal manera que para ejecutar cada uno se deberá de lanzar la siguiente instrucción en un terminal:(prueba1 en caso de la primera prueba, y prueba4 en caso de la cuarta prueba).

```
carmelo00@ubuntu:~$ source robot/devel/setup.bash
carmelo00@ubuntu:~$ roslaunch mirobot prueba1.launcher
```

Ilustración 39: Lanzar Robot

Una vez lanzada esta instrucción nos abrirá el simulador, con el mundo tal y como le necesitamos para realizar la prueba, ya solo tenemos que introducir velocidad a las uniones de forma manual en el menú de la derecha.

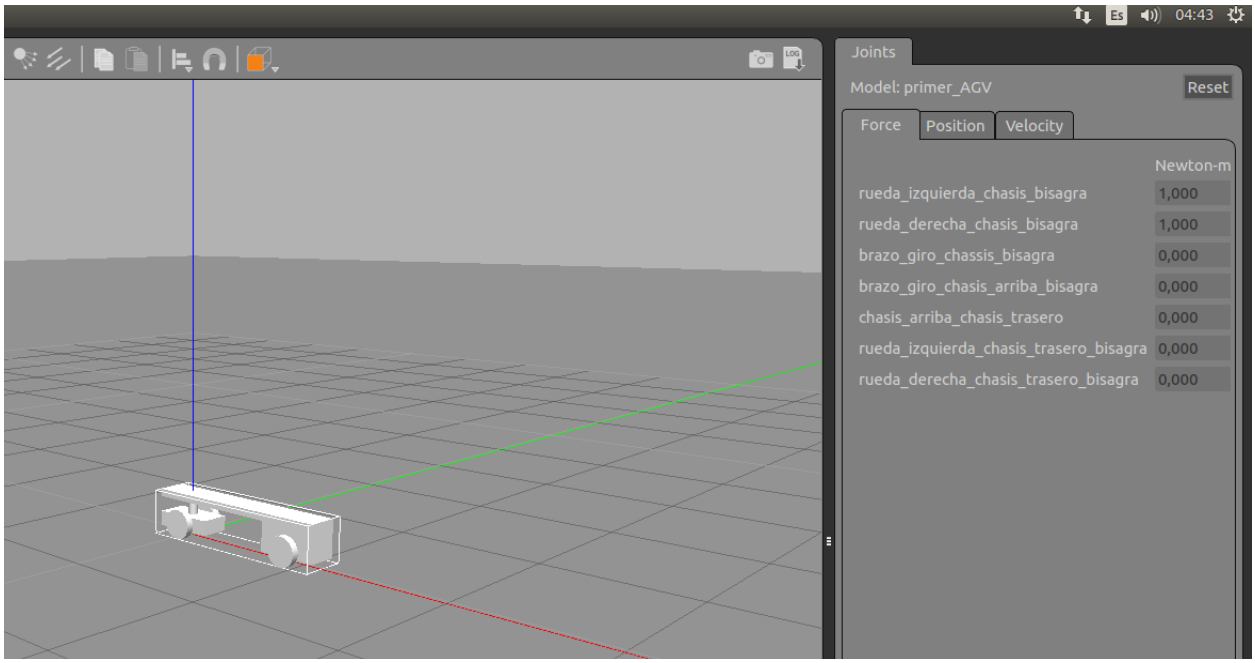


Ilustración 40: Valores fuerzas

3.2. Pruebas con Plugins

Para ejecutar las pruebas será sencillo, aunque algo más complejo que las anteriores, estas 5 pruebas tienen como antes su lanzador:

- Insertar Agv y probar cambio dirección se corresponde con prueba5.launcher

Modelo de robot	Modelo 4 ruedas
Objetivo	Insertar robot en Gazebo y probar cambios de dirección del robot
Resultado	El robot se comporta de manera normal, se mueve a derecha e izquierda según se lo ordenamos. Se observa como realiza los giros cuando lee los ficheros de cambio de dirección

Tabla 30: Prueba 5

- Mover detener y retroceder se corresponde con prueba6.launcher

Modelo de robot	Modelo de robot de 2 ruedas con sensor
Objetivo	Ver comportamiento del robot ante los diferentes objetos que tiene alrededor
Resultado	Se puede observar como el sensor detecta los objetos, y va recalculando según se acerca a ellos, el robot responde de manera perfecta a todas las ordenes, se para y continua según le indicamos

Tabla 31: Prueba 6

- Mover un objeto después de moverse el robot con brazo se corresponde con prueba7.launcher

Modelo de robot	Modelo Gazebo
Objetivo	Ver como podemos mandar diferentes ordenes a diferentes puntos del robot
Resultado	Se aprecia como el robor mueve por una parte sus ruedas, y por otro su brazo robótico, se para delante de un obstáculo, y mueve su brazo robótico para tirarlo

Tabla 32: Prueba 7

- Mover dos robots se corresponde con prueba8.launcher

Modelo de robot	Modelo 2 ruedas y modelo Gazebo
Objetivo	Visualizar como puede incluir nuestro mundo 2 robots y cada uno recibir sus propias ordenes
Resultado	Se visualiza como cada robot recibe sus ordenes, y se comporta tal y como se desea

Tabla 33: Prueba 8

De tal manera que para ejecutar cada uno deberemos de lanzar la siguiente instrucción en un terminal:

```
carmelo00@ubuntu:~$ source robot/devel/setup.bash
carmelo00@ubuntu:~$ roslaunch mirobot prueba1.launcher
```

Ilustración 41: Lanzar prueba

En un nuevo terminal lanzaremos un nodo de ROS para que nos muestre los parámetros de velocidad que tienen las uniones:

```
carmelo00@ubuntu:~$ source robot/devel/setup.bash
carmelo00@ubuntu:~$ rostopic  echo -c /miRobot_m
```

Ilustración 42: Lanzar publicador

Y en otro nuevo terminal lanzaremos un nodo de ROS para que nos mueva los robots leyendo los fichero que queramos(en el ejemplo el fichero avanzar, pero podría ser otro fichero):

```
carmelo00@ubuntu:~$ source robot/devel/setup.bash
carmelo00@ubuntu:~$ rostopic pub /mirobot std_msgs/String "e ../robot/src/mirobo
t/programas/avanzar"
```

Ilustración 43: Lanzar suscriptor

Esto será común a todas estas pruebas, aunque cada una ejecutará un fichero diferente, e irán cambiando en función de la movilidad deseada, se detallará más esta parte en un vídeo explicativo adjunto a la entrega.

3.3. Fallos de programación

Hemos detallado 3 fallos significativos que también se explicarán con vídeo para una mejor interpretación.

- No podemos representar una situación de viento, o gravedad de fuerza lateral al robot, porque la idea era darle una inercia de manera manual, y una fuerza recibida por ROS opuesta, pero se sobreponen entre ambas.
- Si se ejerce una fuerza desproporcionada sobre el objeto, este sale disparado por los aires, de manera muy poco parecida a la realidad.
- Si nos falta unir una rueda a su eje, o la unimos a otro punto del propio robot, la rueda se comporta de manera muy diferente a la realidad, no es que se quede desplazada, sino que afecta a la trayectoria del vehículo de manera poco convincente.

VI - INFORME

Se desarrollará un pequeño informe sobre la viabilidad de incorporar la tecnología ROS-Gazebo en una empresa de robótica.

1. Introducción

Los antecedentes de este proyecto se sitúan en el departamento de I+D de la empresa ASTI donde surgió la posibilidad de incorporar simuladores de entorno los cuales serían capaces de realizar tareas de inspección y mantenimiento de una manera más sencilla y eficiente.

La introducción de un simulador podría facilitar en gran medida todas las tareas que se desarrollan en la compañía. Ya sea desde el punto de las pruebas de programación, así como prevenir errores, inspeccionar máquinas...

El simulador propuesto es Gazebo, que destaca por tener un motor de renderizado avanzado, soporte para plugins, programación en la nube, un enorme repositorio con la mayoría de robots comerciales y una extensa gama de sensores y cámaras.

Como complemento ideal de este simulador, aparece el sistema operativo para robots ROS el cual está creciendo a un ritmo trepidante. Cada vez más y más investigadores de todo el mundo se están sirviendo de las ventajas que ROS ofrece para sus propios proyectos. De hecho, según algunas fuentes, solo en 2016 se han destinado más de 150 millones de dólares como capital de riesgo en negocios fundamentados en ROS, lo cual dice mucho sobre la relevancia de este sistema operativo.

La viabilidad de llevar este avance tecnológico a cabo depende de las dificultades que pueda haber en el desarrollo de esta tecnología, las dificultades de adaptarlo a una empresa, así como de lo costoso que es aprender un sistema tan innovador y del que no se tienen muchas referencias.

En este informe con lo aprendido durante estos cuatro meses, se intentará ver los puntos más relevantes, para posteriormente evaluar su viabilidad y apostar por ello de manera futura.

Ilustración 44: Visitantes de ROS por países

2. Aspectos relevantes

2.1. Objetivos

El objetivo principal de este informe es valorar la viabilidad de incorporar el sistema operativo ROS y un simulador de entornos como podría ser Gazebo en una empresa de logística interna. Los puntos más destacables de incorporar estas tecnologías son:

- Trabajar sin necesidad de tener un entorno similar al del cliente: Podría darse el caso de que un robot para un cliente tuviese que actuar sobre unas condiciones físicas extremas, y quizás no tengamos medios suficientes para poder poner dichas condiciones, mientras que en un simulador sería más sencillo.
- Evitar posibles daños: Los robots podrían tener daños por una mala programación o una inclemencia del entorno
- Trabajar sin necesidad de tener el robot: No es necesario contar con el robot para poder trabajar con el, solo hay que tenerlo diseñado en el simulador y trabajará.
- Mejorar el trato con el cliente: se podría simular al momento el entorno en función de las necesidades del cliente, de tal manera que se podrían ir mandando versiones del simulador sobre como se va avanzando.

Por lo cual se puede decir que el objetivo es que en un periodo de tiempo de 6 meses, los programadores hayan diseñado todos los robots existentes en la empresa, así como incorporado los sensores necesarios, y se pueda empezar a hacer pruebas validas sobre los robots.

2.2. Metodología

Inicialmente se instala un sistema operativo Ubuntu, dentro del gestor de máquinas virtuales VMWare, una vez instalado el sistema operativo se descarga e instala el simulador Gazebo, en el cuál se desarrollaran todas las pruebas que se realizan.

Las primeras pruebas que se realizan fueron sobre el funcionamiento del propio simulador, se completaron tutoriales[6] sobre como construir tu propio robot, construir tu propio mundo, incluir sensores, incluir plugins etc

Una vez se adquieren los conocimientos necesarios se pasa al desarrollo de diferentes modelos de robots que más tarde serán claves para desarrollar un mundo propio, en este punto aparece SDF, lenguaje en el que se construyen los robots y los mundos, interpretado por Gazebo.[7]

En este punto, una vez se aprende a crear diferentes robots, se crean 3 modelos diferentes, con diferentes características, uno tiene una parte móvil, otro un sensor, otro unicamente 2 ruedas...

Para lanzar un mundo se vio que era necesario ROS, un framework para el desarrollo de software para robots, cuya finalidad es inicialmente es lanzar nuestro mundo con nuestro robot dentro de el en el simulador, comunicado con un nodo Máster.

Una vez entendido el funcionamiento de los nodos ROS, buscamos que nuestro robot hable, para lo cual se desarrolla un primer programa que se compila con catkin y se incluye en un plugin del robot, nuestro robot dirá “Hola Mundo” cuando es lanzado.

Ahora nuestra próxima meta es conseguir que nuestros robots se muevan por medio de instrucciones lanzadas en otros nodos, para ello con la ayuda de Raúl Trapiela experto en ROS se construye una librería al compilar el código en la cual podemos suscribirnos y publicar en diferentes nodos, de esta manera el robot nos puede decir que partes a las que darles velocidad tiene, y el programador puede ordenar moverse al robot desde un nodo diferente.

Una vez realizado este aprendizaje pasamos a la realización de diferentes pruebas que se han pensado para poder comprobar el comportamiento de los robots en el simulador, y se saca una conclusión sobre su comportamiento en comparación con un robot real.

Finalmente, se pasa a desarrollar este informe, en el cuál inicialmente se fijaron los puntos con el tutor, para más adelante ir completando según íbamos investigando los demás puntos.

2.3. Contenidos

En la implantación de estos sistemas, enumeraremos los puntos a tener en cuenta:

- Preparación del entorno
- Simulación de robots
- Creación de robots
- Comunicación con robot(ROS)
- Pruebas con robots creados

Para todo ello usaremos un sistema operativo ubuntu, un simulador Gazebo,un sistema operativo para el robot (ROS), y un visualizador de fuerzas RVIZ. Estas serían las herramientas principales a usar.

2.4. Temporalidad

Según la experiencia que se ha adquirido en este proyecto podemos decir que en 5 meses de trabajo se ha aprendido a desarrollar modelos con SDF, y manejar con fluidez el simulador Gazebo, así como de saber manejar nodos ROS. Teniendo en cuenta que el principio es lo más complejo, contando con que tenemos una buena base en este proyecto con la que acortar los plazos, y que se trabajará en un grupo por lo que se avanzará colectivamente, podríamos estimar que serían 2 meses de aprendizaje para poder adquirir los conocimientos que aquí se exponen.

Más tarde habría que centrar más tiempo en manejar todas las librerías y funcionalidades de ROS, el cual no sabríamos estimar.

En segundo lugar estaría el tiempo necesario de implantación, que dependerá de si se busca una implantación total , o parcial (en todos los robots y departamentos de la empresa o solo en algunos). Estimamos que dada nuestra experiencia para una implantación total, desarrollar todos los modelos de robots dentro del simulador, y manejar el simulador serián 6 meses de trabajo, en un grupo de 5 programadores.

2.5. Metodología de implantación

Inicialmente la metodología de implantación para todos los programadores sería aprender con el material facilitado en este TFG los aspectos básicos, para así contar con unos conocimientos sobre el entorno a utilizar, y su funcionamiento que sirvan de base.

Más tarde sería conveniente crear un grupo de trabajo y así poder ir avanzando colectivamente en los diseños, ir realizando pruebas, para más adelante ir incorporando robots en el simulador.

Finalmente se deberá conseguir un lenguaje de comunicación con el robot por medio de ROS y comprobar que la programación del robot real y el simulado es similar ante la misma programación.

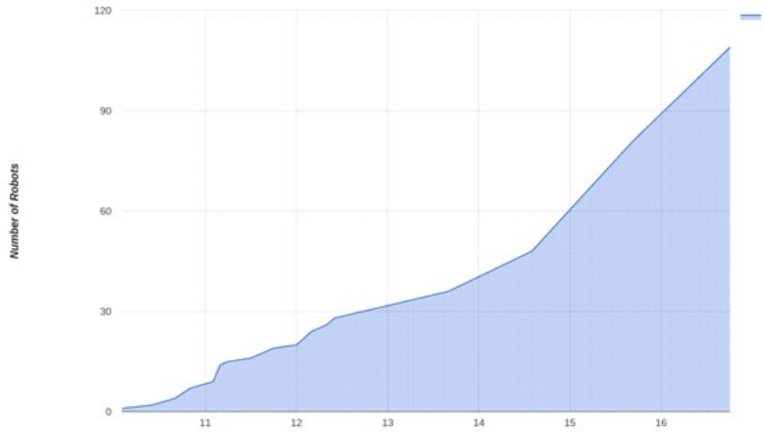


Ilustración 45: Número de robots creados por año

3. Estudio Mercado

3.1. Empresas del sector

3.1.A. JBT

Es una empresa de soluciones tecnológicas diseñan diferentes tipos de AGVs, para empresas, hospitales, farmacéuticas...

Según se puede apreciar en su página web trabajan con simuladores similares a Gazebo, como vemos en la siguiente ilustración, por lo que podríamos decir que un simulador sería ventajoso para este tipo de empresas.

Podemos saber más sobre está empresa en su página web: www.jbtc-agv.com

Ilustración 46: Simulador de JBT

3.1.B. Egemin

Dematic Egemin es un proveedor de vanguardia de los sistemas estándar de vehículos automatizados guiados (sistemas AGV) para aplicaciones específicas.

Cuentan con un software propio para la navegabilidad y la gestión de los AGV, por lo que no tenemos información de si está basado en ROS o no.

Podemos saber más sobre está empresa en su página web:

http://www.egemin-automation.com/en/

3.1.C. Electric-80

Es una empresa enfocada en producir AGVs para las empresas de bienes de consumo de movimiento rápido, especialmente en los sectores de alimentos, bebidas y tejidos. Buscan mejorar constantemente sus conocimientos tecnológicos, de ingeniería e informática para optimizar la gestión de todos los procesos.

Tanto es así que según su página web podemos ver que ofrecen al cliente servicios de simulación para diseñar su entorno, como podemos ver en la siguiente imagen.

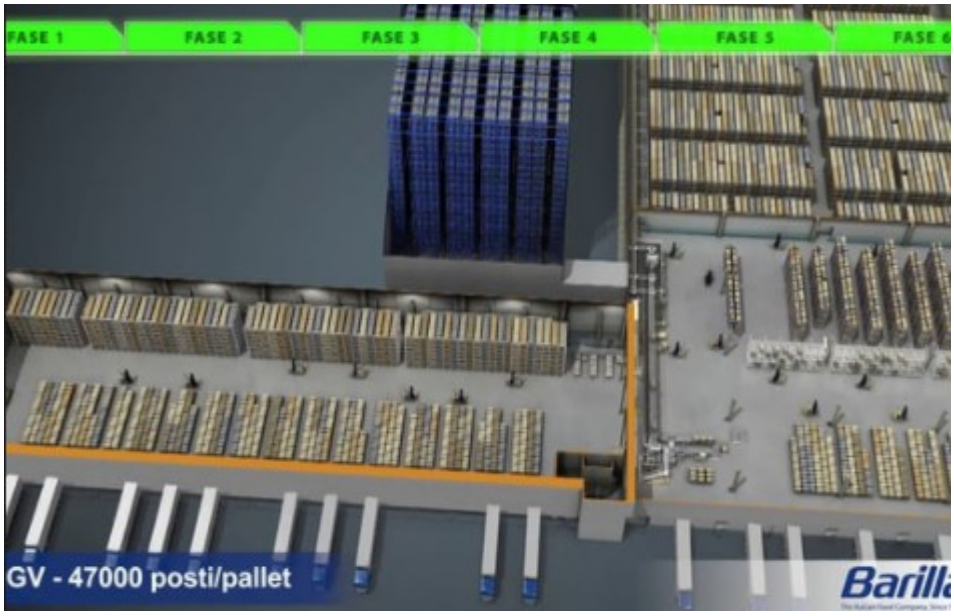


Ilustración 47: Simulador de Electric-80

Podemos saber más sobre está empresa en su página web:

http://www.elettric80.com/it/

3.2. Otros

Hemos encontrado otras empresas que usan la tecnología que deseamos implantar para sus negocios, vamos a enumerarlas:

- erle robotics: Ejemplo de un empresa que se dedica al mundo de la programación de drones y trabaja con ROS.

Ilustración 48: Erle robotics

- Diferentes start-ups focalizadas en robótica móvil, se encargan de poner en contacto a empresarios y programadores para trabajar, en algunas soluciones que han desarrollado se puede ver el uso de ROS. http://www.startupedmonton.com/
- Difetentes grupos de investigación que desarrollan sobre ROS.

4. Recursos humanos

La idea sería contar con un grupo de trabajo de investigación para poder ir avanzando de manera conjunta en el mundo ROS, siendo importante la cooperación entre los integrantes de este grupo para el aprendizaje en estas tecnologías.

El grupo podría contar con 5 personas que se adjudicarían principalmente de avanzar en el mundo del modelado con Gazebo, más adelante se encargarían nuevas funciones a este grupo.

Ilustración 49: Puntos en los que se crean robots

4.1. Conocimientos previos necesarios

En el grupo de trabajo, todos deberían ser titulados en grado de ingeniería informática, o en su defecto podrían ser ingenieros con un máster en algún ámbito relacionado con la programación. Pero se necesitaría un alto grado de nivel como programador.

5. Plan financiero

5.1. Costes económicos de licencias y hardware

Herramienta	**Programa**	**Coste**
Sistema Operativo anfitrión	Linux	0€
Sistema operativo robótico	ROS	0€
Simulador	Gazbebo	0€
IDE	Eclipse	0€
Ordenador	Ordenador de trabajadores	5*750€= 3750€
TOTAL:		**3750€**

5.2. Costes económicos de trabajadores

Según las estimaciones que hemos podido hacer, creemos que en 6 meses el equipo de trabajo ya tendría un nivel alto en el manejo de estos programas, además se ha estimado que el sueldo de cada persona sea de 17€/hora, y decidimos con anterioridad que el grupo de trabajo sería de 5 personas, por lo que:

6 meses son unas 18 semanas a 40 horas semanales son unas 720 horas.

720 horas a 17€ la hora son 12240€.

Serían 5 los trabajadores destinados al grupo de trabajo, por lo que 12240€ por 5 trabajadores serían 61200€

En seguridad social habría que pagar aproximadamente un 30% de lo que ingresan adicionalmente, por que finalmente serian 79560€

5.3. Previsión de nuevos gastos

En cuanto a los cálculos que se han hecho de implantación no se ha tenido en cuenta que los ordenadores asignados al equipo de desarrollo tienen una vida útil de 4 años, por lo que podremos decir que los nuevos gastos serán en un periodo aproximado de 5 años de:

5 equipos * 750€ por equipo 3750€

Es decir, cada 5 años se deberá gastar unos 3750€ en nuevos equipos para los programadores.

5.4. Previsión de ingresos y amortización

Teniendo en cuenta que Javier Miguélez, ingeniero de desarrollo de negocio de ASTI dice que el AGV: “a partir de 12000€ puede ser lo caro que quieras” [8] vamos a suponer que un coste medio de un AGV será de 20000€, por lo cual sería amortizado a partir del cuarto AGV que se venda.

Es difícil estimar cuanto tiempo se gana con la programación en un simulador pero estimaremos que los robots se programan ahora más rápido, y que se producen menos errores en la programación que ocasionen revisiones. Es por ello que podremos ahorrar unos 3000€ de cada robot entre fallos y tiempo perdido, así que para amortizar los 83310€ (3750€ + 79560€) suponiendo que vendemos 4, 7, 13, 22 robots del primer al cuarto año respectivamente.

A continuación he realizado unos cálculos para estimar los ingresos y la amortización a través de dos fórmulas: el VAN y la TIR.

VAN (Valor actual neto):

Amortización 7,5%

$VAN = Di - [(Cobros - pagos): (1+K)^1] + [(Cobros - pagos): (1+K)^2] + [(Cobros - pagos): (1+K)_3] + [(Cobros - pagos): (1+K)_4]$

$VAN = 83310 - [(20000-17000)*4:(1+0,075)^1] + [(20000-17000)*7:(1+0,075)^2] + [(20000-17000)*13:(1+0,075)^3] + [(20000-17000)*22:(1+0,075)^4] = 11162,79+18172,4+31401,9+49438,2 = 26864,29$

Hay que tener en cuenta que el desembolso inicial es elevado. El primer año las ventas son mínimas, y van aumentando poco a poco hasta llegar al cuarto año que es cuando se empieza a obtener beneficios (26864,29€). De aquí en adelante todo lo que se obtenga será en positivo.

TIR (Tasa interna de retorno):

	Robots vendidos	Ingresos	Gastos	Beneficio	
Coste inicial			83310	-83310	
Primer año	4	80000	68000	12000	
Segundo año	7	140000	119000	21000	
Tercer año	13	260000	221000	39000	
Cuarto año	22	440000	374000	66000	
				TIR (2 años)	-42%
				TIR (3 años)	-6%
				TIR (4años)	18%

Ilustración 50: TIR

Definimos TIR como el interés producido por un determinado proyecto o inversión que tiene pagos (valores negativos) e ingresos (valores positivos) que se dan en períodos regulares. Según vemos en la ilustración 50 el TIR comienza a ser rentable a partir del cuarto año, y amortizaríamos la inversión entre el tercer y el cuarto año, cuando el TIR sea igual al 0%.

6. Conclusión

Después de realizar este pequeño informe abordando los puntos más importantes a tener en cuenta en la implantación de estas tecnologías en una empresa, finalizo afirmando que es altamente ventajoso el uso de una simulación en los comportamientos de los AGVs.

Según se ha podido ver, otras empresas del sector ya usan herramientas similares para la simulación de sus entornos, de tal manera que sería ventajoso para cualquier empresa del sector implantar la tecnología.

Gazebo es un simulador de múltiples robots para espacios abiertos, es capaz de simular conjuntos de robots, sensores y objetos, en un mundo tridimensional, y se combina de manera idónea con ROS.

Como se ha observado en diferentes gráficas, el sistema ROS está siendo usado cada día por más personas y empresas, y su irrupción en el mundo de la robótica es una realidad puesto que cada vez más empresas apuestan por él.

En cuanto al precio de implantación he de decir que es no es un precio, sino una inversión a medio-largo plazo, puesto que solo habría que realizar una inversión inicial para pagar al grupo de trabajo durante 6 meses (donde no serán productivos), pero luego los gastos serían insignificantes.

Viendo los resultados del VAN y el TIR, podemos estimar que a partir del cuarto año la inversión sería rentable, por lo que se recomiendo apostar por estas nuevas tecnologías a largo plazo.

Por último, desde un punto de vista personal y después de haber estado trabajando estos programas, viendo su buen funcionamiento y las ventajas que podría dar a una empresa de este tipo, se aconseja basándome en este informe su implantación.

VII - REFERENCIAS

Bibliografía

[1] Colaboradores de la Wikipedia, Copyleft, 2017,
[2] Colaboradores de la wikipedia, GNU General Public Licence, 2017,
[3] Colaboradores de la Wikipedia, Información de <<Licencia BSD>>, 2017,
[4] Colaboradores de la Wikipedia, Apache License, 2017,
[5] Gurley, Gabriel: «A Conceptual Guide to OpenOffice.org 2 for Windows and Linux» 2007
[6] Gazebo, Gazebo Tutorials, 2014,
[7] SDF: «Specification» 2014
[8] Soraya Paniagua, ASTI, la pyme española líder europea en Vehículos de Guiado Automático, 3 de julio del 2017, 06:13 UTC,

Impreso en Burgos el domingo, 17 de septiembre de 2017